Parallel Time and Proof Complexity

Klaus Aehlig

Abstract

Consider the following variant of quantified propositional logic. A new, parallel extension rule is introduced. This rule is aware of independence of the introduced variables. The obtained calculus has the property that heights of derivations correspond to heights of Boolean circuits. Adding an uninterpreted predicate on bit-strings, akin to an oracle in relativised complexity classes, this statement can be made precise. Consider proofs of the statement that a given circuit can be evaluated. The most shallow of these proofs in the said calculus has a height that is, up to an additive constant, the height of the circuit considered.

The main tool for showing lower bounds on proof heights is a variant of an iteration principle studied by Takeuti. This reformulation might be of independent interest, as it allows for polynomial size formulae in the relativised language that require proofs of exponential height.

An arithmetical formulation of the iteration principle yields a strength measure for theories in the language of relativised two-sorted Bounded Arithmetic. This measure provides all the separations Dynamic Ordinal Analysis provides, but extends to theories where the latter fails to produce any separation, due to the overhead of first-order (i.e., sharply-bounded) cut elimination.

The new measure can also be used to investigate relativised theories for small complexity classes that are not necessarily based on restricted forms of induction. In particular, it can be used for theories that axiomatise computations, most prominently, deterministic and non-deterministic space computations, and the evaluation of circuits.

Before studying relativised versions of these theories it is necessary to define relativised versions of the corresponding complexity classes. New definitions of relativised complexity classes within polynomial time are introduced. As opposed to the existing definitions, they have the property that all known inclusions are preserved, and that every complexity class is closed under composition.

Contents

1 Introduction and Related Work

This thesis is about using methods of mathematical logic to study problems in computational complexity.

Even though the most prominent measures of complexity are time and space, quite early in the history of complexity theory, logical concepts have been used to describe computation. For example, the polynomial-time hierarchy [59] essentially is a hierarchy of quantifier alternations. It serves as a complexity-theoretic analogue of the arithmetical hierarchy of degrees of undecidability. It extends Fagin's characterisation [23] of NP as the set of problems definable in second-order existential logic.

The main focus of this thesis are complexity classes that are "small" in the sense that they are contained within polynomial time. These classes are getting increasingly more attention, as, with the increase in computational power, bigger and bigger problems can be solved; with the size of problems manageable nowadays, just stating that a task can be solved in time "polynomial in the size of the input" is too coarse a notion. It is often also of importance whether a problem can be efficiently parallelised.

In this thesis we mainly investigate the complexity classes L, NL, AC^k, and NC^k with particular emphasis on computation relative to an oracle. For these relativised classes we show full separation. Despite not being a new result, the method chosen has several benefits, and results of independent interest are shown on the way. For example, we will obtain all separations by a uniform principle. Also, by linking to formal theories, we open up the possibility to also calibrate mathematical arguments on this computation-related scale.

Bounded Arithmetic as a Link Between Computational Complexity and Mathematical Logic

Bounded Arithmetic is a link between computational complexity and mathematical logic. The object of study are formal arithmetical theories, usually in some form or another restrictions of Peano Arithmetic, and therefore objects well-studied in mathematical logic. On the other hand, we can study their definable functions: Consider a theory T and assume that T proves a statement of the form $\forall x \exists y \varphi(x, y)$. This statement asserts the existence of a (Skolem) function. Assuming the theory is sound, such a function has to exist. So one can ask which complexity is needed to compute a possible Skolem function—if there is a computable one at all. Now varying φ over a (syntactically defined) class $\mathcal{F}$ of formulae, one obtains the $\mathcal{F}$-definable functions of T.

Particularly interesting are choices of T and $\mathcal{F}$ where the $\mathcal{F}$-definable functions of T form some well-known complexity class. The first example of such theories was introduced by Buss [10]. He introduced a hierarchy S_2^i of first-order theories where the Σ_{i+1}^b-definable functions of S_2^{i+1} are precisely [10] the functions in the polynomial-time closure of the i'th level of the polynomial-time hierarchy [59]. Moreover, some form of converse holds as well [40]. The hierarchy of theories S_2^i collapses, i.e., $\mathsf{S}_2 = \bigcup_i \mathsf{S}_2^i$ is finitely axiomatisable, if and only if the polynomial-time hierarchy collapses provably in S_2.

Recently, theories related to other complexity classes, including small ones, have been developed as well [14, 15, 16, 18, 19]. They are best presented [66] in the setting of finite model theory [33]. In other words, rather than building on standard first-order logic, they use a setting akin to second-order arithmetic. The objects of discourse belong to one of two sorts. One of them is the sort of natural numbers. We should think of this sort as ranging over indices into some data structure. The main data objects, and also the input to functions, is represented by the second sort, that of finite sets. Finite sets, in this context, are most naturally thought of as bit-strings. The elements of the set describe which bits are set in the represented string. Such a multi-sorted approach also extends well [58] to much bigger complexity classes like PSPACE.

Having a connection between formal theories and computational complexity can serve several purposes. On a mathematical side, one can take well-known mathematical theorems, like a discrete version of the Jordan curve theorem, and study [47] the computational complexity inherent in this theorem. The way this is achieved is by identifying the smallest theory (of a set of theories strongly linked to complexity in the above-described sense) in which this theorem is provable. The notion of "smallest" can be made more precise than just failing to prove it in any smaller theory; one can actually show that all the axioms of this smallest theory can, over some very weak base theory, be reobtained by just taking the theorem in question as an axiom. This style of connecting mathematical statements and computational complexity is known as "low-level reverse mathematics" [46].

On the computational complexity side, one can use this connection to take advantage of the tools of mathematical logic to learn about the complexity classes in question. If it can be shown that some $\mathcal{F}$-definable functions of T are not $\mathcal{F}$-definable in some other theory T', then the complexity classes corresponding to the $\mathcal{F}$-definable functions of T and T' are separated. Areas of mathematical logic that are suitable to separate formal arithmetical theories include model theory and proof theory.

The Proof-Theoretic Method

In this thesis we use a traditional proof-theoretic approach to study formal theories and, ultimately, show that certain statements are not derivable in some formal theory. Proofs in arithmetical systems are translated into propositional logic and the possible heights of these proofs are studied.

In traditional ordinal-informative proof theory, an arithmetical proof is translated into a single propositional proof with infinitely branching rules and transfinite (but well-founded) height. This proof is then transformed into a proof with the same propositional formula as conclusion but some desirable structural properties. Usually one demands that the transformed proof be cut-free, or that at most cuts of a certain logical form be present. Depending on the arithmetical theory we started with, we can find an ordinal as a bound on the height of the transformed translated proof. If this bound is tight, then this ordinal is the so-called "proof-theoretic ordinal" of that theory.

On the other hand, there are statements A_α such that any cut-free proof of (the propositional translation of) A_α in propositional logic requires a proof of height at least the ordinal α. The most prominent such statement is the principle of transfinite induction up to α. Now, if α is bigger than the proof theoretic ordinal of that theory, we have an explicit example of a statement not provable in the given theory.

In the context of Bounded Arithmetic one considers arithmetical proofs of formulae $A(x)$ with a distinguished free variable x. They are then translated into families of propositional proofs [50] of the propositional statements $A(\underline{n})$ for $n \in \mathbb{N}$. Each individual proof is of finite height and contains only finitely-branching rules; the role of the transfinite height is taken over by the growth of the height of these proofs [6], as depending on n.

In order to show unprovability results, we need some substitute for what the principle of transfinite induction was in ordinal-informative proof theory. In other words, we need a formula $A(x)$ such that any (possibly non-uniform) family of proofs for (the propositional translations of) $A(\underline{n})$ for $n \in \mathbb{N}$ has a height growing fast with n. One of the contributions of this thesis is to provide such a principle with a clear computational meaning: the sequential iteration principle. This (true) principle states that for a function $F\colon [2^n] \to [2^n]$ the function can be iterated, i.e., the value $F^{f(n)}(0)$ exists. "Existence" is formulated by asserting that there is some y such that $(f(n), y)$ is in the graph of the iteration $\ell \mapsto F^\ell(0)$, where that graph is axiomatised in the obvious step-by-step manner. Here $f(n) \in \{0, 1, \ldots 2^n - 1\}$ is some number depending on n. Propositional proofs of this statement will require a height of $f(n)$. Now, if f is sufficiently fast growing we obtain the desired unprovability results.

Iteration as a Sequential Principle

The iteration principle has a computational meaning in terms of computation time. Intuitively, one can only start evaluating $F(F(0))$ once $F(0)$ has been obtained. Similarly, only once the computation of $F(F(0))$ is completed, one can start computing $F(F(F(0)))$, and so on. So iteration is inherently sequential and therefore a good indicator of how much time a computation can use up. Note that for this intuition to hold true, it is essential that the domain and range of F are sufficiently big—otherwise, one could exploit parallelism to precompute F at all possible values. In our formulation of this principle, domain and range are exponentially big. This suffices, as we only consider models of computations where the amount of steps done in parallel is polynomially bounded.

To formally justify our intuition, we study the height of a circuit needed to compute the $F^{(\ell)}(0)$. It will turn out (Proposition 4.1.8 and Theorem 4.1.9) that this height is precisely ℓ. So we have found our link to parallel computation, as circuits are generally considered an appropriate model of parallel computation [12, 60, 13] where the height coincides with computation time.

In that way, the iteration principle allows one to assign to every complexity class—and, in fact, even to every mathematical theory—the amount of sequentiality available. For complexity classes this is just another, however interesting, scale on which to compare them. For other mathematical theories, e.g., those based on induction principles, this is a precise way to understand the computational power inherent in their arguing principles.

Relativised Complexity

In the iteration principle we used a function $F\colon [2^n] \to [2^n]$. Such an object is given by $n \cdot 2^n$ many bits. In other words, an exponential amount of information is needed to describe such an object. That much information cannot be stored by the complexity classes we consider, and it cannot be given as input in the usual sense either. So we need other means of talking about F.

On the side of arithmetical theories we follow standard proof-theoretic practise and add an uninterpreted relation symbol. Note that a predicate of strings of length $n\log(n)$ contains precisely $n \cdot 2^n$ many bits of information.

On the side of computation, a predicate on strings of roughly the size of the input usually is given as an oracle. We think of an oracle as a device where a computed string is given as a question and a yes-or-no answer is returned. The future computation may (and usually will) depend on the answer, including further questions asked. The oracle, however, is committed

to give an answer only depending on the question; it may not depend on the history of questions.

There is a well-established theory of computation relative to an oracle for the complexity classes polynomial time and bigger. These complexity classes have in common that the machine can store the question to be asked to the oracle. For classes like logarithmic space, this is no longer the case. So some extra care has to taken when defining relativised computation for small complexity classes. There do exist some approaches in the literature [64, 65], but none of them is satisfactory in the sense that they preserve the known inclusions and closure properties.

This thesis will therefore, following a joint article with Stephen Cook and Phuong Nguyen [4], introduce new definitions. These new definitions of small complexity classes relative to an oracle preserve the known inclusions, and every complexity class is closed under composition. Such a definition did not exist before. Moreover, NL^{α} will turn out to be closed under complement and contained in deterministic space $\mathcal{O}(\log^2(\mathfrak{n}))$.

Propositional Logic

As explained, a major part of a proof theoretic analysis is working in propositional logic. Doing cut-elimination for a sufficiently involved calculus of infinitary propositional logic can be a major task—even more if the height of the transformed proof has to be optimal.

This thesis devotes a non-negligible part to studying a calculus for relativised propositional logic. On the one hand, developing a suitable calculus and studying its properties is a necessity for the proof-theoretic study of the theories for the complexity classes we are interested in. On the other hand, this study is of independent interest. It will turn out that the heights of proofs in our propositional calculus, called AC^0-Tait, have a tight connection to boolean circuits. More precisely, consider a (relativised) boolean circuit and the formal statement (in propositional logic) that it can be evaluated. This statement will be provable in AC^0-Tait. However, every such proof will require a height that is at least equal to that of the circuit. This result was first presented in a joint article with Arold Beckmann [2].

In this way, we have a logical calculus that is strongly related to parallel computation. Such a connection might become another approach of a better understanding the complexity of parallel computation from a theoretical point of view. In any case, using semantics closely related to machine models of computation has proven useful already [1].

Outline of Contents

This thesis is organised as follows. As all our discourse is about computation relative to an oracle, we first (in Section 2) investigate, following a joint article with Stephen Cook and Phuong Nguyen [4], a sensible notion of relative computation for small complexity classes. It seems that no completely satisfactory definition has been given in the literature [64, 65] before.

Then, a calculus for propositional logic, AC^0-Tait, is introduced (in Section 3). It is shown (in Section 4) that proofs in this calculus have a tight connection to circuit heights.

We then introduce (in Section 5) various theories of Bounded Arithmetic and study (in Section 6) their propositional translation into the said calculus. Computing (in Section 7) the "sequential strength" shows that it yields the expected heights, that is, that of the corresponding circuit. We also note that the "sequential strength" shows the same picture for the theories defined via restricted induction, as does the "dynamic ordinal" [7]. Moreover, it does so without suffering the blindness of the latter in small growth rates. "Dynamic ordinal analysis" requires the polynomial (i.e., poly-logarithmic in the value) overhead of first-order cut elimination. This overhead is not needed to compute the "sequential strength". Partial cut elimination suffices for the Boundedness Lemma for AC^0-Tait (Theorem 4.2.17). So our new measure has all the benefits of the existing one, but additionally strictly extends it. It also has the additional advantage of having a clear complexity-theoretic meaning in the realm of boolean circuits.

2 Relativised Complexity Classes

Computational complexity classifies problems according to the amount of computational resources (like space, (parallel) time, and similar) that suffice to solve them. Absolute separations of most computational complexity classes currently seem out of reach. However, by relativising computation to an oracle as additional parameter most questions of separation can be settled—even though not in a unique way [5].

In the present work, we mainly think of the oracle as an additional, very large, input to the computation. Very often we consider problems where the task is to find out some property of the oracle. Therefore it will make sense to consider problems without input in the traditional sense.

The purpose of this section, which follows work by Aehlig, Cook, and Nguyen [4], is to formally define a relativised notion for various models of computation and to introduce a sensible notion of relativised versions of known small complexity classes. This is not a trivial task, as the complexity classes ought to be closed under composition and known inclusions ought to relativise.

2.1 Preliminaries

Notation 2.1.1. By $\mathbb{N} = \{0, 1, 2, \ldots\}$ we denote the set of natural numbers.

Notation 2.1.2. If n is a natural number, we write $[n]$ as an abbreviation for the set $\{0, 1, \ldots, n-1\}$. That is, in set theoretic terms, $[n] = n$.

Notation 2.1.3. If A and B are sets we denote by $f\colon A \to B$ that f is a (total) function from A to B. In other words, $f \subset A \times B$, such that for each $x \in A$ there is precisely one $y \in B$ such that $\langle x, y\rangle \in f$. We write $f(x) = y$ to denote that $\langle x, y\rangle \in f$.

If f is a function, by $\mathsf{dom}(f) = \{x \mid \exists y.\langle x, y\rangle \in f\}$ we denote its domain and by $\mathsf{rng}(f) = \{f(x) \mid x \in \mathsf{dom}(f)\}$ its range.

Notation 2.1.4. If A and B are sets we denote by $f\colon A \rightharpoonup B$ that f is a partial function from A to B. In other words, f is a function, its domain $\mathsf{dom}(f)$ is a subset of A and its range $\mathsf{rng}(f)$ is a subset of B.

Definition 2.1.5 ($\mathfrak{P}(A)$)**.** If A is a set, then by $\mathfrak{P}(A) = \{B \mid B \subset A\}$ we denote the power set of A.

Notation 2.1.6. We use $\mathfrak{f}$, $\mathfrak{g}$, and $\mathfrak{h}$ to range over (arbitrary) functions from the natural numbers to the natural numbers.

Notation 2.1.7 ($\mathfrak{n}$)**.** We use $\mathfrak{n}$ to denote the identity on the natural numbers; that is, $\mathfrak{n}(n) = n$.

Notation 2.1.8 ($\mathfrak{f} + \mathfrak{g}$, $\mathfrak{f} \cdot \mathfrak{g}$, $\mathfrak{f}^{\mathfrak{g}}$)**.** Sum, product, and exponentiation of functions are defined to be the point-wise sum, product, or exponentiation. In other words, $(\mathfrak{f} + \mathfrak{g})(n) = \mathfrak{f}(n) + \mathfrak{g}(n)$, $(\mathfrak{f} \cdot \mathfrak{g})(n) = \mathfrak{f}(n) \cdot \mathfrak{g}(n)$, and $(\mathfrak{f}^{\mathfrak{g}})(n) = (\mathfrak{f}(n))^{\mathfrak{g}(n)}$.

Notation 2.1.9 ($\mathfrak{f}(\mathfrak{g})$)**.** By $\mathfrak{f}(\mathfrak{g})$ we denote the composition $\mathfrak{f} \circ \mathfrak{g}$ of the functions $\mathfrak{f}$ and $\mathfrak{g}$. So $\mathfrak{f}(\mathfrak{g})(n) = \mathfrak{f}(\mathfrak{g}(n))$. In particular, $\mathfrak{f}(\mathfrak{n})$ is the same as $\mathfrak{f}$.

Definition 2.1.10 ($f^{(k)}$)**.** If $f\colon A \to A$ is a function from a set into itself and $k \in \mathbb{N}$ a natural number, we define the k'th iterate $f^{(k)}$ of f by induction on k as follows.

$f^{(0)}$ is the identity on A, that is, $f^{(0)}(a) = a$. Moreover, $f^{(k+1)} = f \circ f^{(k)}$, that is $f^{(k+1)}(a) = f(f^{(k)}(a))$.

Notation 2.1.11. If there is no risk of confusion with the pointwise operations, we write f^k as a shorthand for $f^{(k)}$.

Notation 2.1.12. If we use a natural number in a context where a function is expected, we identify this natural number with the corresponding constant function. For example $(2\mathfrak{n} + 1)(n) = 2n + 1$.

Definition 2.1.13 ($\mathfrak{f} \leq \mathfrak{g}$)**.** Functions are partially ordered by the point-wise order, that is, we write $\mathfrak{f} \leq \mathfrak{g}$ to denote $\forall n(\mathfrak{f}(n) \leq \mathfrak{g}(g))$.

Definition 2.1.14 ($\mathfrak{f} \leq^{\mathrm{e}} \mathfrak{g}$)**.** By $\mathfrak{f} \leq^{\mathrm{e}} \mathfrak{g}$ we denote that $\mathfrak{g}$ eventually dominates $\mathfrak{f}$. In other words, $\mathfrak{f} \leq^{\mathrm{e}} \mathfrak{g}$ if and only if $\exists N \forall n \geq N(\mathfrak{f}(n) \leq \mathfrak{g}(n))$.

Notation 2.1.15. We also use Notations 2.1.8 and 2.1.12, and Definitions 2.1.13 and 2.1.14 for functions from the naturals to the non-negative reals.

Remark 2.1.16. It is a well known fact, that the set $\{\mathfrak{n}^c + c \mid c \in \mathbb{N}\}$ is confinal in the set of polynomial functions; in other words, for every polynomial function $\mathfrak{p}$ there is a $c \in \mathbb{N}$ such that $\mathfrak{p} \leq \mathfrak{n}^c + c$.

Notation 2.1.17. We use $\mathfrak{p}, \mathfrak{q}$ to range over polynomial functions.

Definition 2.1.18 ($\mathcal{O}(\mathfrak{f})$)**.** If $\mathfrak{f}$ is a function from $\mathbb{N}$ to $\mathbb{N}$ then we define the set $\mathcal{O}(\mathfrak{f})$ of functions of growth at most $\mathfrak{f}$ as follows.

$$\mathcal{O}(\mathfrak{f}) = \{\mathfrak{g} \mid \exists C \in \mathbb{N}.\, \mathfrak{g} \leq^{\mathrm{e}} C \cdot \mathfrak{f}\}$$

Remark 2.1.19. Unfolding Definition 2.1.18 we get

$$\mathcal{O}(\mathfrak{f}) = \{\mathfrak{g} \mid \exists C \in \mathbb{N} \exists N \in \mathbb{N} \forall n \geq N (\mathfrak{g}(n) \leq C \cdot \mathfrak{f}(n))\} .$$

Remark 2.1.20. If $\mathfrak{f} \geq 1$ then

$$\mathcal{O}(\mathfrak{f}) = \{\mathfrak{g} \mid \exists C . \, \mathfrak{g} \leq C \cdot \mathfrak{f}\} .$$

Proof. Let $\mathfrak{g} \in \mathcal{O}(\mathfrak{f})$ and let N, C be as in Definition 2.1.18. Let M be the maximum of the "exceptional values", that is, set $M = \max\{\mathfrak{g}(n) \mid n \leq N\}$. Then for all $n \in \mathbb{N}$ we have $\mathfrak{g}(n) \leq (C + M) \cdot \mathfrak{f}(n)$; in deed, for $n < N$, we have $\mathfrak{g}(n) \leq M \leq (C + M) \leq (C + M) \cdot \mathfrak{f}(n)$ since $\mathfrak{f}(n) \geq 1$, and for $n \geq N$ we have $\mathfrak{g}(n) \leq C \cdot \mathfrak{g}(n) \leq (C + M) \cdot \mathfrak{g}(n)$. □

Notation 2.1.21 ($\mathfrak{f} + M$, $M \cdot \mathfrak{g}$, $\mathfrak{f}^M$)**.** We extend the pointwise operations (Notation 2.1.8) to operations on sets in the obvious way, tacitly adding a downwards closure with respect to eventual domination. That is, if M is a set of functions from $\mathbb{N}$ to $\mathbb{N}$, then we define the following.

$$\begin{array}{lcl} \mathfrak{f} + M & = & \{\mathfrak{h} \mid \exists \mathfrak{g} \in M . \, \mathfrak{h} \leq^{\mathrm{e}} \mathfrak{f} + \mathfrak{g}\} \\ M \cdot \mathfrak{f} & = & \{\mathfrak{h} \mid \exists \mathfrak{g} \in M . \, \mathfrak{h} \leq^{\mathrm{e}} \mathfrak{g} \cdot \mathfrak{f}\} \\ \mathfrak{f}^M & = & \{\mathfrak{h} \mid \exists \mathfrak{g} \in M . \, \mathfrak{h} \leq^{\mathrm{e}} \mathfrak{f}^{\mathfrak{g}}\} \end{array}$$

Remark 2.1.22. Continuing Remark 2.1.16 we observe that the polynomially bounded functions are precisely the functions $\mathfrak{n}^{\mathcal{O}(1)}$.

Definition 2.1.23 (log)**.** We use the following approximation of the binary logarithm as a function from $\mathbb{N}$ to $\mathbb{N}$.

$$\log(n) = \min\{k \mid 2^k \geq n\}$$

Proposition 2.1.24. log *is a monotone function and* $n \leq 2^{\log(n)}$.

Proof. Immediate from our Definition 2.1.23. □

Proposition 2.1.25. $\log(2n) \leq \log(n) + 1$

Proof. By Proposition 2.1.24 we know $n \leq 2^{\log(n)}$, so $2n \leq 2 \cdot 2^{\log(n)} = 2^{\log(n)+1}$, hence $\log(n) + 1$ is an upper bound on $\log(2n)$. □

Proposition 2.1.26. $\log(2^k) = k$

Proof. Trivially $2^k \geq 2^k$. Moreover, for $k' < k$ we have $2^{k'} < 2^k$. So k is indeed the smallest n with $2^n \geq 2^k$. In other words, $\log(2^k) = k$. □

Proposition 2.1.27. *If* $n \geq 2$ *then* $2^{\log(n)} < 2n$.

Proof. Let $k = \log(n)$. Since $n \geq 2$ we know that $\log(k) > 0$ as $2^0 = 1 < 2$. From the minimality of k we know that $2^{k-1} < n$, hence $2^k = 2 \cdot 2^{k-1} < 2n$. □

Corollary 2.1.28. $2^{\log} \in \mathcal{O}(\mathfrak{n})$

Proof. This is a consequence of Proposition 2.1.27. □

Lemma 2.1.29. $\log(n^c) \leq c \log n$

Proof. $n \leq 2^{\log n}$ by Proposition 2.1.24. Hence $n^c \leq (2^{\log n})^c = 2^{c \log n}$. So $c \log n \geq \min\{k \mid 2^k \geq n^c\} = \log(n^c)$. □

Corollary 2.1.30. $\log(\mathfrak{n}^c) \in \mathcal{O}(\log(\mathfrak{n}))$

Proof. This is an immediate consequence of Lemma 2.1.29. □

Lemma 2.1.31. *Let $n \geq 2$ and $c \in \mathbb{N}$. If $n \geq 2^c$ then $c \log n \leq \log(n^{c+1})$.*

Proof. Let $k = \log(n)$. Then k is *minimal* such that $2^k \geq n$ and therefore $2^{k-1} < n$; note that by our assumption $n \geq 2$ we know that $k \geq 1$ and therefore $k-1$ is a well-defined natural number. We get $2^{kc-c} = (2^{k-1})^c < n^c$, so $2^{kc} = 2^c \cdot 2^{kc-c} < 2^c n^c \leq n \cdot n^c = n^{c+1}$. Therefore, $c \log n = kc < \log(n^{c+1})$. □

Corollary 2.1.32. $\log(\mathfrak{n}^{\mathcal{O}(1)}) = \mathcal{O}(\log(\mathfrak{n}))$

Proof. The inclusion $\log(\mathfrak{n}^{\mathcal{O}(1)}) \subseteq \mathcal{O}(\log(\mathfrak{n}))$ is a consequence of Lemma 2.1.29. The inclusion $\mathcal{O}(\log(\mathfrak{n})) \subseteq \log(\mathfrak{n}^{\mathcal{O}(1)})$ follows from Lemma 2.1.31. □

Definition 2.1.33 (2_k)**.** By induction on k we define a function $2_k \colon \mathbb{N} \to \mathbb{N}$ as follows. $2_0 = \mathfrak{n}$ and $2_{k+1} = 2^{2_k}$.

Notation 2.1.34. If A is a set, then by A^* we denote the set of all finite sequences over elements of A. The empty sequence is denoted by ε. The set of all finite sequences of length n over elements of A is denoted by A^n.

Example 2.1.35. $\emptyset^* = \{\varepsilon\}$

Definition 2.1.36 (Language over A)**.** The *languages over A* are defined to be the subsets of A^*.

Remark 2.1.37. Obviously, $A^* = \bigcup_{n \in \mathbb{N}} A^n$ and this union is disjoint. This is a useful principle to define functions with domain A^*; they can be defined by defining separate functions for each input length. This is, for example, done in Definition 2.4.3.

Definition 2.1.38 (Strings Coding a Natural Number)**.** We say that $w = a_\ell \dots a_1 a_0 \in \{0,1\}^*$ *codes* $n \in \mathbb{N}$ if $n = \sum_{i=0}^{\ell} a_i 2^i$.

Example 2.1.39. ε, 0, 00, 000, ... are all codes of the natural number 0. Possible codes of the natural number 1 are 1, 01, 001, and so on.

Proposition 2.1.40. *If $i \in [n]$ then there is precisely one string $\underline{i}$ of length $\log(n)$ that codes i.*

Proof. By Proposition 2.1.24 we know that $i < n \leq 2^{\log(n)}$. Hence there is a code of that lengths. Uniqueness follows by the fact that different binary representations of a number only differ in leading 0s. □

Notation 2.1.41. If the underlying size parameter n is understood from the context, we use $\underline{j}$ to denote the unique string of length $\log(n)$ coding j that is shown to exist in Proposition 2.1.40.

Also, if n is understood from the context, we identify strings over $\{0,1\}$ or strings of truth values (reading "true" as 1 and "false" as 0) with natural numbers they code according to Definition 2.1.38 and vice versa.

Remark 2.1.42. Notation 2.1.41 should not be confused with the notation $\underline{i}$ of numerals in arithmetical theories, introduced in Definition 5.1.10. As, however, we refrain from using Notation 2.1.41 within formulae in the language $\mathcal{L}_2(\alpha)$ of Bounded Arithmetic (see Definition 5.1.1) no confusion can arise.

Definition 2.1.43. If Σ is a set with $0, 1 \in \Sigma$, and if $A, B \subset \Sigma^*$ are languages over Σ, then their *join* $A \oplus B$ is defined to be

$$A \oplus B = \{0w \mid w \in A\} \cup \{1w \mid w \in B\} .$$

2.2 The Dynamic Aspect of Complexity

Computational complexity is concerned with the question of how much the "difficulty" of a problem grows (in the sense of resources needed, like time or space) if the size of the problem grows. So, all the problems computational complexity is concerned with have an underlying "size parameter". In the same way as the notion of a random variable appropriately "hides" the underlying probability space we develop in this subsection a general notion of a "dynamic object" that appropriately hides the underlying size parameter.

Definition 2.2.1 (Sized Set)**.** A *sized set* is a pair $(S, |\cdot|)$ consisting of a set S and a *size function* $|\cdot| \colon S \to \mathbb{N}$ from the set to the natural numbers.

Notation 2.2.2. If the size function $|\cdot|$ is understood from the context, we also call S a sized set, expressing that we're talking about the sized set $(S, |\cdot|)$.

Definition 2.2.3 (Dynamic S-Object)**.** If S is a sized set then a *dynamic S-object* is a function $a\colon \mathbb{N} \to S$. Its *growth rate* $|a|$ is the function $|a|\colon \mathbb{N} \to \mathbb{N}$, $n \mapsto |a(n)|$.

Notation 2.2.4. Following usual conventions, we call a function $a\colon \mathbb{N} \to S$ also a *family* if S-objects and write a_n as a shorthand for $a(n)$. In particular we use the notations $a\colon \mathbb{N} \to S$ and $(a_n)_{n\in\mathbb{N}}$ synonymous, if it is understood that $a_n \in S$.

Definition 2.2.5 (Polynomial Size)**.** A dynamic object a is said to be of *polynomial size*, if its growth rate is polynomially bounded, that is, if $|a| \in \mathfrak{n}^{\mathcal{O}(1)}$.

Definition 2.2.6 (Constant Size)**.** A dynamic object a is said to be of *constant size*, if its growth rate is. That is, a is of constant size if $|a| \in \mathcal{O}(1)$.

Definition 2.2.7 (Exponential Size)**.** A dynamic object a is said to be of *at least exponential size*, if its growth rate dominates some exponentially growing function, that is, if for some $\varepsilon > 0$ it is the case that $(1+\varepsilon)^{\mathfrak{n}} \leq^{\mathrm{e}} |a|$.

Remark 2.2.8. When working with small complexity classes, there is hardly any benefit in knowing that some measure is bounded by an exponentially growing function. Therefore, we will use the expression "exponential size" to mean "at least exponential size", if there is no danger of confusion.

Notation 2.2.9. When speaking of polynomial or exponential size objects we tacitly assume that these are dynamic objects. For example, a "polynomial size formula" is a dynamic formula of polynomial size; in particular, being a dynamic formula, a "polynomial size formula" is not a single formula, but a sequence of formulae.

Relations between objects are lifted to their dynamic versions pointwise. For example, if in Corollary 4.2.19 we will say that some "polynomial size formula requires exponential height proofs" we claim that a dynamic formula, that is, a sequence $(A_n)_{n\in\mathbb{N}}$ of formulae exists that has two properties. The first claim is that $(\mathsf{sz}(A_n))_{n\in\mathbb{N}} \in \mathfrak{n}^{\mathcal{O}(1)}$, for the size $\mathsf{sz}(A_n)$ that will be introduced in Definition 3.1.12. The second claim is that for every sequence $(d_n)_{n\in\mathbb{N}}$ of proofs such that for every n the proof d_n proves A_n there is some $\varepsilon > 0$ such that for sufficiently large n the height of d_n is at least $(1+\varepsilon)^n$.

Definition 2.2.10 (Object parametrised by an oracle)**.** By an *C-object, parametrised by an oracle over Σ*, we mean a function $a\colon \mathfrak{P}(\Sigma^*) \to C$. We just call it an *object, parametrised by an oracle* if Σ is understood. If a is an object parametrised by an oracle, and $A \subseteq \Sigma^*$, by "a, for parameter A" we refer to $a(A)$.

2.3 Relativised Circuits

The most direct way to express a boolean function is by a propositional formula. Boolean circuits can be thought of as propositional formulae with sharing; identical subparts have to be computed only once, as intermediate results can be used several times and at several places.

Circuits seem a good model of parallel computation. Other models of parallel computation can be expressed well by means of restricted families of circuits [12]. The connection to parallel time is that independent parts of a circuit can be evaluated simultaneously, whereas gates depending on each other have to be evaluated one after another.

In fact, in this work we will use circuits as our gold standard for the notion of parallel time. A problem is considered to be highly sequential, if all small (i.e., sub-exponential) circuits solving it are of big height.

Relativised circuits have also been considered by Wilson [64, 65] as a model for relativised parallel computation.

Definition 2.3.1 (Circuit)**.** A *circuit* over a set V of variables is a finite, vertex-labelled, directed acyclic graph together with

- a repetition-free list of elements of V, called the "input list", and
- a list of nodes, possibly with repetitions, called the "output list",

where every vertex of the graph is of one of the following types, i.e., labelled as described.

- An "input node", labelled with an element of $x \in V$ or with $\neg x$, where $x \in V$. We require input nodes to have no incoming edges and x to occur in the list of inputs.
- An "and-gate", labelled with the symbol $\bigwedge$, assumed not to occur in V.
- An "or-gate", labelled with the symbol $\bigvee$, assumed not to occur in V.
- A "not-gate", labelled with the symbol $\neg$, assumed not to occur in V. Not-gates are required to have precisely one incoming edge.

- An "oracle gate", labelled with the symbol α, assumed not to occur in V, and a list, possibly with repetitions, of vertices where an edge to this oracle gate exists. The length of this list is called the *width* of the oracle gate.

- A "negated oracle gate", labelled with the symbol $\bar{\alpha}$ and a list, possibly with repetitions, of vertices where an edge to this oracle gate exists. Again, the length of this list is called the width of the oracle gate.

The vertices of a circuit are also called its *nodes* or *gates*. The *size* is the number of its nodes.

The *level* of a node is the maximum of the lengths of all paths (starting at an arbitrary node and) ending in this node or 0 if the node has no incoming edges. The *height* is the smallest natural number that is strictly greater than the levels of all nodes.

Remark 2.3.2. Given that a circuit is acyclic (and finite), every path has finite length. In particular, the notions of level and height are well defined.

Example 2.3.3. The empty circuit has height 0. And the circuit consisting of a single input node (and no output nodes) has height 1 and its only node is at level 0.

Definition 2.3.4 (Variable Assignment)**.** A *variable assignment* for a set V of variables is a mapping $\eta\colon V \to \{0,1\}$.

Definition 2.3.5. An *oracle* is a set $A \subset \{0,1\}^*$ of strings over $\{0,1\}$.

Definition 2.3.6 (Circuit Evaluation)**.** A *circuit evaluation* of a circuit C over variables V, relative to a variable assignment η for V, and an oracle A is a mapping $v\colon N \to \{0,1\}$ of the nodes N of the circuit to $\{0,1\}$ with the following properties.

- If $n \in N$ is an input node labelled x, then $v(n) = \eta(x)$. If $n \in N$ is an input node labelled $\neg x$, then $v(n) = 1$ if and only if $\eta(x) = 0$.

- If $n \in N$ is an and-gate, then $v(n) = 1$ if and only if for all nodes n' such that there is an edge from n' to n it holds that $v(n') = n$.

- If $n \in N$ is an or-gate, then $v(n) = 1$ if and only if there is a node n' with an edge from n' to n such that $v(n') = 1$.

- If $n \in N$ is a not-gate then $v(n) = 1$ if and only if $v(n') = 0$ for the (uniquely determined) node n' with an edge to n.

- If $n \in N$ is an oracle gate labelled $\alpha n_1 \dots n_k$ then $v(n) = 1$ if and only if $v(n_1) \dots v(n_k) \in A$, that is, if the string built from the values of the associated list of inputs belongs to the oracle.
- If $n \in N$ is a negated oracle gate labelled $\alpha n_1 \dots n_k$ then $v(n) = 0$ if and only if $v(n_1) \dots v(n_k) \in A$, that is, if the string build from the values of the associated list of inputs belongs to the oracle.

Proposition 2.3.7. *For every circuit C over variables V, variable assignment η for V, and oracle A there is precisely one circuit evaluation of C relative to η and A.*

Proof. By induction on the natural number h we prove that there is one and only one mapping v from all nodes of C of level strictly smaller than h to $\{0, 1\}$ satisfying all the conditions of a circuit evaluation that only refer to nodes of level strictly smaller than h. Note that by taking h the height of the circuit we obtain the claim.

For $h = 0$ there is nothing to show, as there are no nodes of level strictly below 0. For the inductive step, the conditions of a circuit evaluation leave one and only one choice on how to extend v. □

Remark 2.3.8. The unique existence shown in Proposition 2.3.7 allows us to speak of *the* evaluation of C under η and A.

Proposition 2.3.9. *Let C be a circuit with input list $\vec{v}$ and A an oracle. If two variable assignments η and η' agree on $\vec{v}$ then the evaluation of C under η and A agrees with the valuation of C under η' and A.*

Proof. By induction on h, we show that the two valuations agree for all nodes of level strictly below h. Taking h to be the height of the circuit gives the claim. □

Definition 2.3.10 (Function Computed by a Circuit)**.** If C is a circuit with input list $v_1, \dots, v_k$ and output list $n_1, \dots, n_\ell$ then it computes, for every oracle A, a function $f_C^A \colon \{0,1\}^k \to \{0,1\}^\ell$, called "the function computed by C relative to A", in the following way. If v is the evaluation of C relative to some, and hence any (Proposition 2.3.9), variable assignment η with $\eta(v_1) \dots \eta(v_k) = w \in \{0,1\}^k$ then $f_C^A(w) = v(n_1) \dots v(n_\ell)$.

Proposition 2.3.11. *For every circuit C, there is another circuit of at most the same height and at most twice the size that does* not *contain any not-gate and computes the same function.*

Proof. By induction on h we show that a circuit C' without not-nodes exists that has at most height h and contains, for every node n of C of level strictly below h, two nodes c_n and c'_n such that for every variable assignment and oracle for the corresponding circuit evaluations v for C and v' for C' it holds that $v(n) = v'(c_n)$ and $v(n) = 1 - v'(c'_n)$. Note that we do *not* require the mapping $n \mapsto c_n$ to be one-one. Also note that taking for h the height of C yields the claim.

For an input node n with label x take c_n to be an input node labelled x and c'_n an input node labelled $\neg x$. For an input node n labelled $\neg x$ take c_n to be labelled $\neg x$ and c'_n labelled x.

For an and-node n with inputs $n_1, \ldots, n_\ell$ take c_n an and-node with inputs $c_{n_1}, \ldots, c_{n_\ell}$ and c'_n an or-node with inputs $c'_{n_1}, \ldots, c'_{n_\ell}$. Similar for or-nodes or oracle nodes.

For a not-node n with input m take $c_n = c'_m$ and $c'_n = c_m$. □

2.4 Circuit Complexity Classes

Following Aehlig, Cook, and Nguyen [4] we now define relativised versions of the AC^k and NC^k complexity classes. The main issue here is how to account for an oracle gate. This is quite obvious for the AC^k-hierarchy, where other gates with unbounded fan-in also count as height and size 1. However, in NC^k gates should have fan-in 2, which, of course, is not possible for oracle gates.

Traditionally [64, 65], an oracle gate with ℓ inputs in an NC^k circuit is considered to be of size ℓ and height $\log(\ell)$. Note that a circuit with binary gates that depends on ℓ inputs has to have at least $\ell - 1$ nodes and height $\log(\ell)$. However, accounting for oracle gates in that way would allow for a combination, in the same circuit, of a deep nesting of small queries and shallowly nested long queries. This seems unnatural and does not fit well with the usual relation to Turing-machine complexity.

We therefore charge, in NC^k, the cost for an oracle gate in the n'th member of the family as height $\log(n)$, regardless of the number of actual number of inputs. Note that for a polynomial size circuit, the number of inputs is bound by a polynomial and $\log(\mathfrak{n}^{\mathcal{O}(1)}) = \mathcal{O}(\log(\mathfrak{n}))$. In other words, our definition allows slightly less nesting than others previously used in the literature.

In Corollary 2.5.5 we will see that these restrictions, together with an appropriate definition of relativised log-space computation, will ensure the known inclusions to be preserved. A careful discussion of the appropriate definition of relativised Turing machine computation, and proofs of the embedding theorems, can be found in Subsection 2.5.

Definition 2.4.1 (Family of Circuits)**.** A *family of circuits* is a family $(C_n)_{n\in\mathbb{N}}$ of circuits in the sense of Definition 2.3.1 such that

- C_n has an input list $v_1, \ldots, v_n$ of length n,
- the number of nodes of C_n grows at most polynomially with n, and
- the maximal width of the oracle gates in C_n grows at most polynomially with n.

A family of circuits is said to be "for a language" if all the C_n have an output list of length precisely one.

Remark 2.4.2. The last condition in Definition 2.4.1 might require some explanation. The general idea of a family of circuits is that every circuit has the correct signature and, moreover, that the n'the circuit is an object of size feasible (i.e., polynomial) in n. For this to work we also have to take "hidden" size into account. In other words, we not only have to ensure that the number of gates grows only polynomially, but also that the information needed to describe a particular gate is polynomially bounded as well. Besides the type of a gate ("and", "or", ...) we also have to remember its incoming edges. For "and" and "or" gates, this information is polynomially bounded in the number of gates of the circuit, as repetitions in the input are not allowed (and wouldn't change the output of the gate anyway). For oracle gates, on the other hand, a complete description of the string to be queried is given. As this string may contain repetitions, a bound on the number of gates of the circuit will not imply a bound on the information needed to describe such a gate. Hence we add the an additional requirement to make the description of the oracle gates feasible.

Definition 2.4.3 (Function or Language Computed by a Family of Circuits)**.** A family $(C_n)_{n\in\mathbb{N}}$ computes a function $\{0,1\}^* \to \{0,1\}^*$, parametrised by an oracle, in the obvious way. That is to say, for a given oracle A, let $f^A_{C_n}\colon \{0,1\}^k \to \{0,1\}^*$ be the function computed by C_n, in the sense of Definition 2.3.10. Then $(C_n)_{n\in\mathbb{N}}$ computes the function $\bigcup_{n\in\mathbb{N}} f^A_{C_n}$.

If $(C_n)_{n\in\mathbb{N}}$ is a family of circuits for a language, we say that $(C_n)_{n\in\mathbb{N}}$ computes the language $\{w \in \{0,1\}^* | f(w) = 1\}$ where $f\colon \{0,1\}^* \to \{0,1\}$ is the function computed by $(C_n)_n$.

Remark 2.4.4. It should be noted that in Definition 2.4.3 the union $\bigcup_{n\in\mathbb{N}} f^A_{C_n}$ is indeed a function again, as the individual circuits of the family have different input lengths.

Definition 2.4.5 ($\mathsf{AC}^k(\alpha)$)**.** For $k \geq 0$, by an $\mathsf{AC}^k(\alpha)$-*circuit* we mean a family $(C_n)_n$ of circuits such that, besides having the properties mentioned in Definition 2.4.1, the height is bounded by $\mathcal{O}((\log(\mathfrak{n}))^k)$.

By $\mathsf{AC}^k(\alpha)$ we denote the set of all functions, parametrised by an oracle, that can be computed by some $\mathsf{AC}^k(\alpha)$-circuit.

Definition 2.4.6 (Oracle Nesting)**.** For every circuit C we say that its *oracle nesting* is the maximal number of oracle gates on any path through C.

Remark 2.4.7. Here we follow the convention that the maximum of the empty set is 0. So a circuit without oracle gates has oracle nesting 0.

Definition 2.4.8 ($\mathsf{NC}^k(\alpha)$)**.** For $k \geq 1$, by an $\mathsf{NC}^k(\alpha)$-*circuit* we mean a family $(C_n)_n$ of circuits such that (besides having the properties mentioned in Definition 2.4.1)

- every and-gate and or-gate has at most two incoming edges,
- the height is bounded by $\mathcal{O}((\log(\mathfrak{n}))^k)$, and
- the oracle nesting is bounded by $\mathcal{O}((\log(\mathfrak{n}))^{k-1})$.

By $\mathsf{NC}^k(\alpha)$ we denote the set of all functions, parametrised by an oracle, that can be computed by some $\mathsf{NC}^k(\alpha)$-circuit.

Note that in Definition 2.4.8 the height restriction is equivalent to the informal description given at the beginning of this subsection (on page 16). Restricting the height to $\mathcal{O}((\log(\mathfrak{n}))^k)$ and the oracle nesting depth to $\mathcal{O}((\log(\mathfrak{n}))^{k-1})$ is the same as to say that the height has to be $\mathcal{O}((\log(\mathfrak{n}))^k)$ where oracle gates are counted as height $\log(\mathfrak{n})$ regardless of their actual width.

Definition 2.4.9 (AC^k, NC^k)**.** An $\mathsf{AC}^k(\alpha)$-circuit or $\mathsf{NC}^k(\alpha)$-circuit is called an AC^k-circuit or NC^k-circuit, respectively, if it does not contain any oracle gates.

Proposition 2.4.10. $\mathsf{NC}^k(\alpha) \subseteq \mathsf{AC}^k(\alpha)$

Proof. Trivial. Just note that every $\mathsf{NC}^k(\alpha)$-circuit also is an $\mathsf{AC}^k(\alpha)$-circuit. □

Proposition 2.4.11. $\mathsf{AC}^k(\alpha) \subseteq \mathsf{NC}^{k+1}(\alpha)$

Proof. Let $(C_n)_{n\in\mathbb{N}}$ be an $\mathsf{AC}^k(\alpha)$-circuit. We have to find an $\mathsf{NC}^{k+1}(\alpha)$-circuit computing the same function. First we note that by the height bound, the oracle nesting of $(C_n)_{n\in\mathbb{N}}$ is bounded by $\mathcal{O}(\log^k)$ and that the size bound implies that number of incoming edges to the gates in $(C_n)_{n\in\mathbb{N}}$ is bounded by $\mathfrak{n}^{\mathcal{O}(1)}$.

Now, let C'_n be obtained by replacing every and-gate or or-gate with ℓ inputs in C_n by a balanced tree of binary and-gates or or-gates, respectively. Then $(C'_n)_{n\in\mathbb{N}}$ obviously computes the same function. As a balanced binary tree with $\ell-1$ nodes has height $\log(\ell)$, the height of C'_n is that of C_n multiplied by a factor $\log(\mathfrak{n}^{\mathcal{O}(1)}) = \mathcal{O}(\log(\mathfrak{n}))$. Therefore $(C'_n)_{n\in\mathbb{N}}$ is an $\mathsf{NC}^{k+1}(\alpha)$-circuit, as desired. □

So far, we have only considered, what is referred to as the "non-uniform" versions of the circuit classes. It is easy to see that they contain non-computable languages. In fact, the full set $\mathfrak{P}(\mathbb{N})$ of real numbers can be encoded into the non-uniform AC^0-languages.

Proposition 2.4.12. *Let $A \subseteq \mathbb{N}$. Then there is an AC^0-circuit computing*

$$\{w \in \{0,1\}^* \mid |w| \in A\} .$$

Proof. We note that for any fixed length n either all strings of length n belong to the language, or they don't. So we can chose C_n to be either the canonical circuit returning 1 on all inputs, or the canonical circuit returning 0 on all inputs, depending on whether $n \in A$. Then $(C_n)_{n\in\mathbb{N}}$ is an AC^0-circuit for the desired language. □

Therefore, one often requires that the members C_n of a circuit family are constructed in a "uniform way", e.g., by only considering those families $(C_n)_{n\in\mathbb{N}}$, where there is some form of program that, given n, computes how C_n looks like.

The most abstract concept of uniformity only exploits the fact, that there are only countably many programs. This is still enough for the typical diagonalisation arguments, as the one in the proof of Theorem 4.1.14.

Definition 2.4.13 (Notion of Uniformity)**.** A *notion of uniformity* is any countable set $\mathcal{U}$ of circuit families.

Definition 2.4.14 (Uniform Circuit Classes)**.** If $\mathcal{U}$ is a notion of uniformity, then $\mathcal{U}$-uniform AC^k, NC^k, $\mathsf{AC}^k(\alpha)$, or $\mathsf{NC}^k(\alpha)$-circuits are those AC^k, NC^k, $\mathsf{AC}^k(\alpha)$, or $\mathsf{NC}^k(\alpha)$-circuits, respectively, that are in $\mathcal{U}$.

Following the tradition of finite model theory, we use first-order definablity as our standard notion of uniformity.

Definition 2.4.15 (First-Order Uniformity)**.** A family of circuits is said to be *first-order uniform*, if there are first-order formulae φ_v, $\varphi_\wedge$, $\varphi_\vee$, $\varphi_\neg$, φ_α, $\varphi_{\bar{\alpha}}$, and $\varphi_{\mathtt{con}}$, in the language $\{0, 1, +, \cdot, <\}$ with the following properties.

The free variables of $\varphi_\wedge$, $\varphi_\vee$, $\varphi_\neg$, φ_α, and $\varphi_{\bar{\alpha}}$ are among $\{x_1, \ldots x_c\}$ if $\mathfrak{n}^c$ is the polynomial bound on the circuit size; the free variables of φ_v are among $\{y, x_1, \ldots, x_c\}$ and those of $\varphi_{\mathtt{con}}$ are among $\{y, x_1, \ldots, x_c, x'_1, \ldots, x'_c\}$.

When evaluated over the finite structure with universe $[n]$, the constants 0 and 1 interpreted by 0 and 1, and $+$ and $\cdot$ interpreted by appropriately restricted addition and multiplication, $\varphi_v(\underline{i}, \underline{a_1}, \ldots, \underline{a_c})$ holds true if and only if, in the n'th circuit of the family, the node numbered by the tuple $(a_1, \ldots, a_c)$ is an input-node for the i'th element of the input list. Similarly, $\varphi_\wedge$, $\varphi_\vee$, $\varphi_\neg$, φ_α, $\varphi_{\bar{\alpha}}$ hold true on $(\underline{a_1}, \ldots, \underline{a_c})$, if and only if the node is an "and gate", "or gate", "not gate", "oracle gate", or "negated oracle gate", respectively. Moreover, $\varphi_{\mathtt{con}}(\underline{i}, \underline{a_1}, \ldots, \underline{a_c}, \underline{b_1}, \ldots, \underline{b_c})$ holds true, if and only if the gate numbered by the tuple $(a_1, \ldots, a_c)$ is the i'th input to the gate numbered by the tuple $(b_1, \ldots, b_c)$.

Remark 2.4.16. Since there are only countably many (tuples) of first-order formulae, first-order uniformity is a notion of uniformity in the sense of Definition 2.4.13.

Definition 2.4.17 (Standard Notion of Uniformity)**.** If, for any circuit class, we speak of its "uniform version" without further specifying a notion of uniformity, we implicitly refer to first-order uniformity.

2.5 Relativised Turing Machines and Their Complexity Classes

Assuming some familiarity with the computational model of a Turing machine [30, 44] we will now discuss how to define computation of a Turing machine relative to an oracle and define the corresponding relativised complexity classes L^α and NL^α for deterministic and non-deterministic logarithmic space, relative to the oracle α.

The need to discuss their definitions arose from the observation that there doesn't seem to be an agreed definition in the literature such that the relativised small complexity classes are closed under composition, let alone AC^0 reductions, and, moreover, the known inclusions

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AC}^1$$

(for the uniform versions of NC^1 and AC^1) are preserved. In fact, Ladner and Lynch [41] show that, for a naive definition of NL^α, a computable oracle α

can be found such that NL^α is not even contained in P^α. The problem with their definition is that the non-deterministic machine is allowed to "guess" oracle queries, rather than computing them. In this introduction, we will give a different example for problems with the naive definition.

We follow the general concept to relativise a Turing machine by means of "oracle tapes". Here, an oracle tape is an additional, write-only tape where the Turing machine can write a question to the oracle; additionally there is an "oracle state" to signal that the query is complete. Once the oracle state is entered, the oracle tape is cleared and the machine changes to a designated "yes" state or a designated "no" state, depending on whether or not the string queried belongs to the language of the oracle.

The first question is how to account for the space used on the oracle tape. For sublinear space classes, like L^α, it seems unreasonable to fully charge the machine for the space used. Indeed, if doing so, L^α wouldn't be, in general, able to compute α as this would require copying the input to the oracle tape. This motivates charging only moderately the space used on the oracle tape, say logarithmically (the space needed to keep track of the head position on the oracle tape) or not at all.

With this moderate charge, a new problem occurs when allowing non-determinism; the oracle might be used as additional, illegitimate storage as the following example shows. Let α be the language consisting of all pairs of a Boolean formula and a satisfying assignment. Clearly, $\alpha \in \mathsf{L}$. But now, for a naive notion of non-determinism, a non-deterministic log-space machine with an oracle for α can solve the satisfiability problem as follows.

> Copy the input to the oracle tape. Then, non-deterministically, write some assignment for the variables on the oracle tape. Query α and accept if and only if the query belongs to α.

However, for any sensible notion of relativisation, an oracle for L shouldn't be of any help to an NL-machine—but an NL-algorithm for the satisfiability problem is not known today. In fact, most computer scientists believe that such an algorithm doesn't exist at all; note that the existence of an NL-algorithm for the satisfiability problem would imply $\mathsf{P} = \mathsf{NP}$. Moreover, there is no obvious way how the L-algorithm for α can be inserted into the above cited NL^α-algorithm as to obtain an NL-algorithm for the same problem.

This obstacle was first noted by Ruzzo, Simon and Tompa [53]. They suggested the restriction that a Turing machine be deterministic unless the oracle tape is empty. The rationale for this restriction is the idea to fully charge the machine for oracle queries, but allow it to use some efficient coding of the query—the deterministic process of writing to the oracle tape is just

the "decoding" of the query implicitly described by the state of the machine in the moment the first symbol is to be written.

The second obstacle we have to face when relativising complexity classes with sub-linear space restriction is the requirement that sensible complexity classes be closed under composition. To see where this obstacle arises, recall the standard proof [34] that L is closed under composition; note that the output of an L-Turing machine, in general, is of polynomial length and therefore cannot be stored entirely by an L-Turing machine. Nevertheless, for M_1 and M_2 Turing machines in L, the composition of their functions can be computed by the following log-space algorithm.

> Start simulating the second Turing machine M_2. If a particular symbol of the output of M_1 is needed, interrupt the current simulation and re-simulate (on a separate section of the work tape) the entire computation of M_1 until the point where the needed symbol is written; other output symbols of M_1 need not be stored.

Even though this algorithm works well for L, a difficulty arises in the case of the relativised version L^α. Consider the cases that M_2 needs the output of M_1 while composing an oracle query. However, if M_1 is in L^α as well it will presumably need to pose oracle questions itself. This shows that, in general, a single oracle tape is not enough.

To overcome this problem, Wilson [64] suggested a stack model for relativised Turing machines. While composing an oracle query, the machine may decide to "push" a fresh tape on the stack of oracle tapes to compose a new query on it; once the new query is answered, the additional tape is "popped" and the answer of this query may be used to continue composing the first query. We observe that a stack of constant height suffices to obtain closure under composition.

Definition 2.5.1 (L^α, NL^α,L, NL)**.** L is the class of all languages computable by a log-space Turing Machine and NL is the class of all languages computable by a non-deterministic log-space Turing Machine

For a unary relation α on strings, L^α is the class of languages computable by log-space, polytime Turing machines using an α-oracle stack whose height is bounded by a constant. NL^α is defined as L^α but the Turing machines are allowed to be non-deterministic when the oracle stack is empty.

It should be noted that every deterministic log-space Turing machine necessarily runs in polynomial time. However, this will become a non-trivial restriction as soon as we consider non-determinism.

The Turing machine classes we defined are already a uniform concept. So, when speaking of its uniform version, we just mean the class itself.

The introduction of a stack of oracle tapes solves both mentioned problems. This is the content of the following proposition and Corollary 2.5.18 which will show that $\mathsf{NL}^{\mathsf{L}^\alpha} = \mathsf{NL}^\alpha$.

Proposition 2.5.2. *L^α is closed under composition.*

Proof. As in the classical proof sketched above (on page 22) the L^α-machine for the composition essentially simulates the second L^α-machine. Whenever output from the first L^α-machine is needed, the computation is interrupted and the first machine is simulated till the needed output is produced. Due to the availability of an oracle stack such a simulation can happen even during the composition of an oracle query—the oracle tapes needed in the simulation of the first machine are just pushed on top of the stack. The height of the oracle stack of the composed machine is bounded by the sum of bounds for the individual machines, and hence bounded by a constant as well. □

Following Aehlig, Cook, and Nguyen [4], we will now show that the inclusion properties that motivated our definition of the relativized circuit complexity classes (Subsection 2.4) as well as that of L^α and NL^α actually hold.

Theorem 2.5.3. *Every language in NL^α can be computed by a uniform family of $\mathsf{AC}^1(\alpha)$ circuits where the nesting depth of the oracle gates is bound by a constant.*

In particular, for the uniform versions, $\mathsf{NL}^\alpha \subseteq \mathsf{AC}^1(\alpha)$.

Proof. We follow the standard proof [48] that $\mathsf{NL} \subseteq \mathsf{AC}^1$, that is, we consider reachability in the graph of configurations of the non-deterministic Turing machine. A small technical problem arises. The contents of the oracle tapes are too big to be considered in the graph—otherwise the graph would no longer be polynomially bounded. Fortunately, the relation "when starting to write on the oracle tape in this particular configuration we can reach a configuration where a 1 is written to the i'th cell of the oracle tape" is a reachability relation. Recall that the machine behaves deterministically if the tape is non-empty. Therefore, the only chance that a situation is reachable where a particular letter is written into a particular cell is that it actually happens. Hence every bit of the oracle query is a reachability relation and therefore AC^1.

More formally, let M be the NL-Turing machine and h the height bound on the oracle stack. A configuration consists of the local state of the machine, the contents of the work tape, the height and the positions of the various Turing machine heads. Note that it does *not* include the contents of the oracle tapes. As the length of the work tape is logarithmically bounded, there are only polynomially many configurations.

For $i = h, h-1, \ldots, 1, 0$ we now consider the reachability relation restricted to those moves where the height of the oracle stack never falls below i. For $i = h$ this is just the normal transition relation of a Turing machine, as no further push operation is possible and any pop operation would result in the height of the oracle stack falling below h. Hence the neighbour relation is a trivial local condition, hence AC^0, and therefore the overall reachability problem is AC^1. (The existence of a path of length at most 2^{k+1} can be described as "there is a node v and paths of length at most 2^k from s to v and from v to t" which is AC^0 in the predicate for paths of length 2^k.)

Now assume that the relation for oracle stack heights of i or bigger is already established as an $\mathsf{AC}^1(\alpha)$-circuit. We consider the corresponding "big-step relation", that is, for configurations with height $i-1$ of the oracle stack where the next transition is a push operation we go directly to the configuration after the corresponding pop. First we note that finding the configuration which performs the corresponding pop is just the reachability relation for configurations with oracle stack never below i. However, the outcome of this pop depends on the answer of the oracle, which, in turn, depends on the question asked. But, as discussed above, every bit of the oracle question is just an instance of the reachability relation for configurations with stack never below i—"can be reach a state where the head on the top oracle tape in position j and the letter 1 is written?". So, a constant height circuit (on top of the already established relation for i) with a single oracle gate suffices for each such big step. Once these big steps are established, reachability is the usual AC^1-problem, just as in the case of $i = h$.

Hence we get an $\mathsf{AC}^1(\alpha)$-circuit for every i. The case $i = 0$ yields the claim. □

Lemma 2.5.4. *For the uniform versions,* $\mathsf{NC}^1(\alpha) \subseteq \mathsf{L}^\alpha$.

Proof. First note, that our the uniformity condition implies that an L^α Turing machine can compute any part of the relevant circuit at will; so we can assume the circuit for the particular input size to be given.

The L^α machine now carries out a depth-first traversal of the circuit. During this traversal, the value of each node visited is calculated, till, at the end of the traversal the value of the output node is computed. It should be noted that during this traversal the only information that has to be remembered is about the current node and the nodes on the path taken from the root to the current node. As $\mathsf{NC}^1(\alpha)$ circuits are of logarithmic height, there are only logarithmically many nodes involved at any one moment. Moreover, for every non-oracle node only a fixed amount of information has to be stored: the direction (left or right) chosen and the value of the sub-node not taken, if already calculated.

By the restriction on $\mathsf{NC}^1(\alpha)$ circuits, there is only a constant number of oracle gates on every path. For every oracle gate, we store the direction taken (which, in general, requires logarithmically many bits); the partially constructed query is stored on an oracle tape. Note that the stack-like access to the oracle tapes fits with depth-first traversal and that, moreover, only a constant-height stack is needed. □

Corollary 2.5.5. *For the uniform versions,* $\mathsf{NC}^1(\alpha) \subseteq \mathsf{L}^\alpha \subseteq \mathsf{NL}^\alpha \subseteq \mathsf{AC}^1(\alpha)$.

Proof. The only non-trivial inclusions are those shown in Theorem 2.5.3 and Lemma 2.5.4. □

Remark 2.5.6 (Relativised Space $\mathcal{O}(\log^k)$)**.** Following the general principle, motivated in this section, of charging oracle queries according to their potential, rather than their actual length, it seems obvious how one should define relativised space $\mathcal{O}(\log^k)$ for $k \geq 1$.

Since Oracle tapes should be charged $\log \mathfrak{n}$, we allow an oracle stack of height $\mathcal{O}((\log \mathfrak{n})^{k-1})$. Of course, we keep the restriction that a non-deterministic machine be deterministic, whenever the oracle stack is not empty. Note that this definition includes our definition of L^α and NL^α as the special case $k = 1$.

Reinspecting the proof of Lemma 2.5.4, one realises that the relativised space classes $\mathcal{O}(\log^k)$ include $\mathsf{NC}^k(\alpha)$. A straight forward generalisation of the proof of Theorem 2.5.3 to these polylogarithmic space classes fails—however not for anything related to the oracles (that would work out correctly!), but simply for the reason that the set of states of such a Turing machine is no longer polynomially bounded.

Definition 2.5.7 (L^α-function, L^α-reductions)**.** A function f from strings to strings is an L^α function, if the lengths of the output is polynomially bounded in the length of the input, and the following problem is in L^α. "Given a string x, a letter c and a natural number i, decide whether $f(x)$ has at least $i+1$ symbols and the i'th symbol of $f(x)$ equals c".

A language L is L^α-reducible to L', if there is an L^α-function f such that $x \in L$ if and only if $f(x) \in L'$. A language L is L^α-complete for a set of languages, if it belongs to the set and every other element of the set is L^α-reducible to it.

Remark 2.5.8. The set of L^α-functions can also be characterised as the set of functions computable by an L^α-machine equipped with an additional write-only output tape.

Proof. To see that an L^α-machine equipped with an output tape can compute any L^α-function, consider the following algorithm. Initialise a counter i to zero. Then repeatedly iterate though all possible symbols c of the output alphabet and decide—using that this problem is in L^α—whether the i'th symbol is c. If so, output c, increment the counter and repeat. If it is found that none of the possible symbols occurs at position i, then the output is ended; just halt in this case.

For the other direction, first note that L^α-machines, having a polynomial time bound, can only write polynomially many symbols. Hence the bound on the output length holds.

Now, an L^α-machine with output tape can be modified in the following way as to solve the decision problem. On input x, i, c, first initialise a counter to 0 and then behave as the original machine, except when an output symbol is to be written. In this case check whether the counter has the value i. If not continue with the counter increased. Otherwise check whether the symbol that is about to be written equals c. Answer the decision problem accordingly. If the original machine halts, then the output string has less than $i + 1$ symbols, so the answer is "no". □

Recall that "s-t-connectivity" is the following problem.

> Given a directed graph and two nodes s and t in it. Decide whether there is a path from s to t.

It is well known [55, 33, 48] that s-t-connectivity is NL-complete under L-reductions, in fact even under AC^0-reductions. Adding the oracle access to our notion of reductions, it remains complete also for NL^α.

Lemma 2.5.9. *The problem s-t-connectivity is complete for NL^α under L^α-reductions.*

Proof. Since s-t-connectivity is in NL, it is trivially in NL^α. As for the other direction, let an NL^α-machine and its input be given. By an L^α-function we can compute, for that particular input, the transition graph of all configurations with empty oracle tapes. Note that there are only polynomially many of these.

Between these configurations the transitions that do not involve oracle tapes can be computed as usual. If a possible transition involves writing to an oracle tape, then the NL^α-machine has to behave deterministically from this moment onwards. Hence the L^α-machine can compute the "big step" till the next time the oracle stack is empty.

Now the question, whether x is in the language of the NL^α-machine is equivalent on whether from the initial configuration the accepting configuration can be reached (we may assume, without loss of generality, that the NL^α-machine clears up its work tape before entering the accepting state). This finishes the proof. □

An important consequence of Lemma 2.5.9 is that NL^α is closed under complement. We actually can appeal directly to Immerman-Szelepcsényi's theorem [32, 61] that NL is closed under complement.

Corollary 2.5.10. NL^α *is closed under complement.*

Proof. Given an NL^α language, we can use the reduction in Lemma 2.5.9 to find, for every x, a graph and two nodes such that there exists a path if and only if x belongs to the language. So a string does not belong to the language, if in the corresponding graph there is no path from s to t. But by Immerman-Szelepcsényi's theorem the complement of s-t-connectivity is in NL as well. This finishes the proof. □

In a similar manner, we obtain a relativised version of Savitch's theorem [54].

Corollary 2.5.11. *Every language in* NL^α *can also be decided by a deterministic oracle Turing machine in space* $\mathcal{O}(\log^2)$.

Proof. By Lemma 2.5.9, every problem in NL^α can, in L^α be reduced to s-t-connectivity. But s-t-connectivity is in NL and hence by Savitch's theorem also in deterministic space $\mathcal{O}(\log^2)$. Therefore the composition of these two procedures is as desired. □

Remark 2.5.12. The proof of Corollary 2.5.10 does not directly generalise to showing that the non-deterministic space classes $\mathcal{O}(\log^k)$, as mentioned in Remark 2.5.6, are closed under complement.

Nevertheless, the idea, also implicit in the proof of Theorem 2.5.3, does generalise. Non-deterministic relativised space $\mathcal{O}(\log^k)$ can be described as reachability in a graph with nodes described by strings of length $\mathcal{O}(\log^k)$ and an edge relation available to a deterministic relativised space $\mathcal{O}(\log^k)$ machine. Now, doing the same "guess and verify the count" algorithm as in the original proof of the Immerman-Szelepcsényi theorem [32, 61] shows that a non-deterministic machine with constantly many pointers (also usable as counters) into this graph can solve non-reachability. This shows the desired closure under complement.

Definition 2.5.13 ($\mathsf{AC}^0(\alpha)$-reduction)**.** A set L is $\mathsf{AC}^0(\alpha)$-reducible to L', if there is a uniform $\mathsf{AC}^0(\alpha)$ circuit whose language relative to $\alpha \oplus L'$ is L.

In other words, L is reducible to L', if L can be computed by a family of polynomial-size constant-height circuits that have, besides the logical gates, gates for α and for L'.

As usual, we call a language $\mathsf{AC}^0(\alpha)$-complete for a complexity class, if every language in the complexity class is $\mathsf{AC}^0(\alpha)$-reducible to that language. Careful analysis of the proof of Theorem 2.5.3 reveals that actually a stronger result is shown there.

Proposition 2.5.14. *The (unrelativised) problem s-t-connectivity is* $\mathsf{AC}^0(\alpha)$*-hard for* NL^α*.*

Proof. As in the proof of Theorem 2.5.3, we follow the standard proof that $\mathsf{NL} \subseteq \mathsf{AC}^1$. That is, we consider the reachability graph of the non-deterministic Turing machine, again not considering the state of the oracle tapes as part of the configuration, but reconstructing it, as needed, by describing it as a reachability relation.

However, as opposed to the proof of Theorem 2.5.3, we use the reachability oracle to solve the reachability problems under consideration, rather than constructing an AC^1-circuit. This will keep the constructed circuit flat, i.e., of constant height. Again, we iteratively solve this problem of the reachability relation restricted to those moves where the height of the oracle stack never falls below i, for $i = h, h-1, \ldots, 1, 0$ where h is the height bound on the oracle stack. □

By a completely similar argument as we obtain

Proposition 2.5.15. *The (unrelativised) line-reachability problem problem is* $\mathsf{AC}^0(\alpha)$*-hard for* L^α

The converse of Proposition 2.5.15 holds as well.

Lemma 2.5.16. L^α *can evaluate* AC^0*-circuits with* α*-gates and gates for line-reachability.*

Proof. As in the proof of Lemma 2.5.4, the L^α machine carries out a depth-first traversal of the circuit. For a line-reachability gate the machine simulates an L-machine solving this problem, computing the inputs as needed (possibly several times, if the L-machine reads a particular bit several times). Since the height of the circuit is constant, the work tapes of the simulated L-machines can be stored in logarithmic space. Moreover, the traversal guarantees that queries to α are only build in a stack-like fashion. □

The proof of Lemma 2.5.16 as presented cannot be directly extended to the evaluation in NL^α of AC^0-circuits with α-gates and gates for s-t-connectivity. It is not known to the author whether this statement actually is true. The problem with a naive extension is that the NL^α machine would have to solve an instance of the s-t-connectivity problem as part of the input for an oracle gate. But in this situation, i.e., while having an oracle query partially constructed, the machine would have to behave deterministically.

Lemma 2.5.17. $\mathsf{L}^{\mathsf{L}^\alpha} = \mathsf{L}^\alpha$

Proof. Since $\alpha \in \mathsf{L}^\alpha$, the inclusion $\mathsf{L}^{\mathsf{L}^\alpha}$ is obvious.

For the other inclusion, let A be an L^α-language and consider an L-Turing-machine with an oracle for A. By Proposition 2.5.15 this machine can be described my an AC^0-circuit with gates for A and line-reachability. Again by Proposition 2.5.15 the language A can be described by an AC^0-circuit with gates for α and line-reachability. Substitution this circuit for the A-gates in the first circuit yields an AC^0-circuit with gates for α and line-reachability. By Lemma 2.5.16 this circuit can be evaluated in L^α. □

Corollary 2.5.18. $\mathsf{NL}^{\mathsf{L}^\alpha} = \mathsf{NL}^\alpha$

Proof. Consider an NL-machine with an oracle in L^α. As long as the oracle stack is empty the machine behaves as an NL machine without oracle. From the moment on, the first symbol is written to an oracle tape, the machine has to behave deterministically. So the computation until the stack is empty for the next time is in $\mathsf{L}^{\mathsf{L}^\alpha}$, which, by Lemma 2.5.17 is L^α. Hence the whole computation can be carried out in NL^α. □

3 Relativised Quantified Propositional Logic

In this section we present a calculus for quantified propositional logic, following Aehlig and Beckmann [2]. It will mainly be in the style of Tait [62], however quantification will be done symbolically, and not by unfolding the (exponentially, but finitely, many) possibilities. The main interest in this calculus will lie in the fact that the height of proofs in this calculus is a meaningful measure and in fact closely related to the height of circuits.

In fact, Theorem 4.3.9 will show (together with Lemma 4.3.3) that, up to an additive constant, for some canonical circuit, the height of the most shallow circuit of sub-exponential size is the height of that very circuit. This canonical circuit will solve an inherently sequential problem as the discussion in Subsection 4.1 will show (where the problem is also formally introduced); the problem considered is to iterate a function that is given by the oracle.

Krajíček and Pudlák [39] studied quantified propositional logic in relation to complexity classes and Bounded Arithmetic. They introduced various dag-like (G_1, G_2,..., G) and tree-like systems (G_1^*, G_2^*,..., G^*). Cook and Morioka (in a slightly modified setting) identified [17] the calculi G_0 and G_0^* which relate to NC^1.

One motivation for the study of restricted propositional proof systems is the relation to weak theories of bounded arithmetic. Some of these theories are introduced in Section 5.

For various complexity classes, corresponding proof systems [51, 57] have been identified. However, a unifying framework for the propositional systems still seems to be missing. We suggest a calculus which is flexible enough to allow for embedding of various theories (as we shall see in Section 6), but is still strict enough that the *height* of proofs is a meaningful measure.

3.1 The Language of Relativised Quantified Propositional Logic

As absolute separation results for complexity classes within polynomial time currently seem out of reach, we have to relativise our language, in order to obtain unconditional separation results. Relativisation in propositional logic consists of adding an uninterpreted predicate (a "parameter") on bit vectors. That is, we allow new propositions of the form $\alpha_k \wp_1 \dots \wp_k$, see Definition 3.1.3.

Definition 3.1.1 (Propositional Atoms)**.** The *atoms of propositional logic* are variables $p, q, r, \dots$, their negations $\bar{p}, \bar{q}, \bar{r}, \dots$, as well as the constants T and F or truth and falsity.

The set of all propositional atoms is denoted by $\mathcal{A}$.

Notation 3.1.2. $\wp$ ranges over elements of $\mathcal{A}$, that is, over the atoms of propositional logic.

The idea of relativisation, that is, the idea of having the possibility of querying an "oracle" (in the sense of relativised computational complexity classes) is modelled by adding a new constructor α to build formulae. This constructor is given a bit string of a fixed length, given at the meta level. In other words, for every k we assume a symbol α_k and a negated symbol $\bar{\alpha}_k$.

We follow the guiding principle of a "call by value" semantics of the oracle. That is, an oracle only receives already computed values and not computations. In (quantified) propositional logic, values correspond to propositional variables and constants, whereas formulae correspond to (special) computations. This intuition motivates the following definition.

Definition 3.1.3 (Propositional Parameter Queries)**.** The *parameter queries of propositional logic* are formulae $\alpha_k \wp_1 \dots \wp_k$ and their negation $\bar{\alpha}_k \wp_1 \dots \wp_k$, where k is a natural number and $\vec{\wp}$ are propositional atoms.

As it is common for Tait-style calculi, conjunction and disjunction are "wide" in the sense that several (in general more than two) formulae are joined together. In this setting, the reasonable notion of quantification is block quantification, that is, we allow to quantify over a whole block of variables at the same time. Note that quantification over a single variable has expressive power of an only binary conjunction or disjunction.

As in the case of the parameter, the lengths of conjuncts, disjuncts and variable blocks to be quantified over comes from the meta level. That is, in the following definition $\bigwedge_1$, $\bigwedge_2$, $\bigwedge_3$ and so on are all different symbols of our language; similar for the other connectives.

Definition 3.1.4 (Quantified Propositional Formulae)**.** The set of *quantified propositional formulae* $A, B, C, \dots$ is built up from the atoms and parameter queries of propositional logic by wide conjunctions $\bigwedge_k A_1 \dots A_k$ and disjunctions $\bigvee_k A_1 \dots A_k$, and block universal $\forall_k p_1 \dots p_k A$ and existential $\exists_k p_1 \dots p_k A$ quantification.

The variables $p_1, \dots, p_k$, as well as their negations $\bar{p}_1, \dots, \bar{p}_k$ are bound in $\forall_k p_1 \dots p_k A$ and $\exists_k p_1 \dots p_k A$. We consider alpha-equivalent formulae as syntactically identical.

Remark 3.1.5. This syntactical identification does not contradict our intuition of syntactical identity as equality of strings of symbols. For example, we could imagine the binding as a meta-syntactical operation [9], replacing

every occurrence of the bound variable by the appropriate de Bruijn index [8]. Then alpha-equal formulae are represented by the identically same string of symbols.

Definition 3.1.6 (Purely Propositional Formulae)**.** A quantified propositional formula without any quantifications is called a *purely propositional formula*.

Notation 3.1.7. We use the term "propositional formula" to mean "quantified propositional formula".

Notation 3.1.8. Syntactical equality is denoted by $\equiv$.

Notation 3.1.9. We write $\wedge$ and $\vee$ for $\bigwedge_2$ and $\bigvee_2$, respectively.

Notation 3.1.10. Even though our official notation is the Polish one, we use $A \wedge B$ and $A \vee B$ as abbreviations for $\wedge AB$ and $\vee AB$, respectively, if there is no danger of confusion. Also, parentheses may be used to facilitate reading or to disambiguate these abbreviations.

Moreover, we write $\alpha_k(\wp_1, \ldots, \wp_k)$ for $\alpha_k \wp_1 \ldots \wp_k$, and $\bar{\alpha}_k(\wp_1, \ldots, \wp_k)$ for $\bar{\alpha}_k \wp_1 \ldots \wp_k$.

Definition 3.1.11. A quantified propositional formula is *α-free*, if it does not contain any propositional parameter α_n, for any n. It is called *closed*, if it does not contain any free propositional variables.

Note that any closed, α-free quantified propositional formula has a standard truth value T or F in the obvious way.

Definition 3.1.12 ($\mathsf{sz}(A)$)**.** If A is a formula of quantified propositional logic, we define its size by induction on A as follows.

- $\mathsf{sz}(\wp) = \mathsf{sz}(\alpha_k \vec{\wp}) = 1$
- $\mathsf{sz}(\bigvee_k \vec{A}) = \mathsf{sz}(\bigwedge_k \vec{A}) = 1 + \sum_{1 \leq i \leq k} \mathsf{sz}(A_i)$
- $\mathsf{sz}(\forall_k \vec{p} A) = \mathsf{sz}(\exists_k \vec{p} A) = 1 + \mathsf{sz}(A)$

Definition 3.1.13 ($\mathsf{dp}(A)$)**.** If A is a formula of quantified propositional logic, we define its depth by induction on A as follows.

- $\mathsf{dp}(\wp) = \mathsf{dp}(\alpha_k \vec{\wp}) = 1$
- $\mathsf{dp}(\bigvee_k \vec{A}) = \mathsf{dp}(\bigwedge_k \vec{A}) = 1 + \max\{\mathsf{dp}(A_i) \mid 1 \leq i \leq k\}$
- $\mathsf{dp}(\forall_k \vec{p} A) = \mathsf{dp}(\exists_k \vec{p} A) = 1 + \mathsf{dp}(A)$

Definition 3.1.14 (Negation)**.** By induction on A a formula $\neg A$ is defined in the obvious way. More precisely, we set $\neg\mathrm{T} \equiv \mathrm{F}$, $\neg\mathrm{F} \equiv \mathrm{T}$, $\neg p \equiv \bar{p}$, $\neg\bar{p} \equiv p$, $\neg(\alpha_k \wp_1 \dots \wp_k) \equiv \bar{\alpha}_k \wp_1 \dots \wp_k$, $\neg(\bar{\alpha}_k \wp_1 \dots \wp_k) \equiv \alpha_k \wp_1 \dots \wp_k$, $\neg(\bigwedge_k A_1 \dots A_k) \equiv \bigvee_k (\neg A_1) \dots (\neg A_k)$, $\neg(\bigvee_k A_1 \dots A_k) \equiv \bigwedge_k (\neg A_1) \dots (\neg A_k)$, $\neg(\forall_k p_1 \dots p_k A) \equiv \exists_k p_1 \dots p_k (\neg A)$, and $\neg(\exists_k p_1 \dots p_k A) \equiv \forall_k p_1 \dots p_k (\neg A)$.

Proposition 3.1.15. $\neg\neg A \equiv A$.

Proof. Induction on A. □

Notation 3.1.16. We use $A \to B$ as abbreviation for $(\neg A) \vee B$ and $A \leftrightarrow B$ as abbreviation for $(A \to B) \wedge (B \to A)$.

Notation 3.1.17. If A is a quantified propositional formula, $\vec{p}$ are pairwise distinct propositional variables, and $\vec{B}$ are quantified propositional formulae, then by $A[\vec{B}/\vec{p}]$ we denote the simultaneous capture-free substitution of all p_i by B_i and of all $\bar{p}_i$ by $\neg B_i$.

Since we identify alpha-equal terms, "capture-free substitution" is a well-defined notion in the sense that the result is uniquely defined (up to our identification). Alternatively, we could assume that we always take a representative where all bound variables are different from all the variables mentioned in the substitution. Note that if we stick to our convention that bound variables are replaced by the appropriate de Bruijn index whereas free variables keep their names, than the naive substitution replacing each occurrence of p_i by B_i and each occurrence of $\bar{p}_i$ by $\neg B_i$ is already capture free.

Notation 3.1.18. When displaying variables of a formula as in $A(\vec{p})$ this should signify that these variables, among others, may occur in A. This notation does not imply that these variables actually do occur free and the list $\vec{p}$ does not necessarily exhaust all the free variables of A. The purpose of such a display of variables is to distinguish certain variables so that later $A(\vec{B})$ can be used as a shorthand for the substitution $A[\vec{B}/\vec{p}]$.

Definition 3.1.19 (Propositional Substitution)**.** A *propositional substitution* is a mapping from propositional variables to quantified boolean formulae that differs from the identity only at finitely many places.

The *range* of a propositional substitution ρ is the set of the image of those variables moved, i.e., the range of ρ is $\{\rho(p) \mid \rho(p) \neq p\}$.

Remark 3.1.20. With substitutions being functions, Definition 3.1.19 is in contradiction with the definition of the range, with substitutions seen as functions (compare Notation 2.1.3). To avoid confusion, we hereby adopt the convention that we'll never apply the range definition for function to substitutions.

We will often call a propositional substitution simply a "substitution", if it is clear from the context, that we speak about propositional logic.

Definition 3.1.21 (Atomic Substitution)**.** A substitution is called atomic, if it maps propositional variables to propositional atoms.

We use σ to range over atomic substitutions.

Definition 3.1.22 (Σ-closure)**.** Let $\mathcal{F}$ be a set of quantified propositional formulae closed under atomic substitutions. The Σ-*closure* $\Sigma\mathcal{F}$ *of* $\mathcal{F}$ is defined to be the smallest set that

- contains $\mathcal{F}$,
- is closed under $\bigvee_k$, that is, if $\vec{A} \in \Sigma\mathcal{F}$ then $\bigvee_k \vec{A} \in \Sigma\mathcal{F}$, and
- is closed under $\exists_k$, that is, if $A(\vec{p}) \in \Sigma\mathcal{F}$ then $\exists_k \vec{p} A(\vec{p}) \in \Sigma\mathcal{F}$.

Note that the notation $\Sigma\mathcal{F}$ implicitly presupposes that $\mathcal{F}$ is closed under atomic substitutions.

Remark 3.1.23. In Definition 3.1.22 we required $\mathcal{F}$ to be closed under renaming of propositional variables, in order to be able to conclude from $\exists_k \vec{p} A(\vec{p}) \in \Sigma\mathcal{F} \setminus \mathcal{F}$ that $A(\vec{p}) \in \Sigma\mathcal{F}$, despite our convention (see Definition 3.1.4) that we identify alpha-equal formulae.

Definition 3.1.24 ($\Sigma_i^q(\alpha)$, $\Pi_i^q(\alpha)$)**.** We define sets $\Sigma_i^q(\alpha)$ and $\Pi_i^q(\alpha)$ by induction on i as follows. $\Sigma_0^q(\alpha) = \Pi_0^q(\alpha)$ is the set of the purely propositional formulae, $\Sigma_{i+1}^q(\alpha) = \Sigma\Pi_i^q(\alpha)$ and $\Pi_{i+1}^q(\alpha) = \{\neg A \mid A \in \Sigma_{i+1}^q(\alpha)\}$.

3.2 Logical Rules in Quantified Propositional Logic

Notation 3.2.1. We use $\Gamma, \Delta, \ldots$ to denote finite sets of formulae.

Definition 3.2.2 (Propositional Rules)**.** The *propositional rules* of quantified propositional logic are the following rules.

$$\frac{}{\Gamma, p, \bar{p}} \qquad \frac{}{\Gamma, \mathrm{T}}$$

$$\frac{}{\Gamma, \alpha_k(\wp_1, \ldots, \wp_k), \bar{\alpha}_k(\wp_1, \ldots, \wp_k)}$$

$$\frac{\Gamma, A_i}{\Gamma, \bigvee_k A_1 \ldots A_k} \qquad \frac{\ldots \quad \Gamma, A_i \quad \ldots \qquad (1 \le i \le k)}{\Gamma, \bigwedge_k A_1 \ldots A_k}$$

Here the $\vec{\wp}$ are arbitrary propositional atoms (Definition 3.1.1). In the last two rules, the formulae $\bigvee_k A_1 \ldots A_k$ and $\bigwedge_k A_1 \ldots A_k$ are called the principal formulae of that inference.

Remark 3.2.3. In all the rules (including those to come) we may always assume without loss of generality that the conclusion is already contained in the premise. For example, a typical instance of the or-rule would in fact be

$$\frac{\Gamma, A_0 \vee A_1, A_i}{\Gamma, A_0 \vee A_1}\,.$$

Definition 3.2.4 (Parameter Extensionality)**.** The *rules of parameter extensionality* are the following rules.

$$\frac{\Gamma, \alpha_k(\wp_1, \ldots, \wp_k) \qquad \ldots \Gamma, \wp_i \leftrightarrow \wp_i' \ldots \qquad (1 \leq i \leq k)}{\Gamma, \alpha_k(\wp_1', \ldots, \wp_k')}$$

$$\frac{\Gamma, \bar{\alpha}_k(\wp_1, \ldots, \wp_k) \qquad \ldots \Gamma, \wp_i \leftrightarrow \wp_i' \ldots \qquad (1 \leq i \leq k)}{\Gamma, \bar{\alpha}_k(\wp_1', \ldots, \wp_k')}$$

Here $\vec{\wp}, \vec{\wp'}$ may be arbitrary atoms of propositional logic.

Definition 3.2.5 (Eigenvariables)**.** At various places we will require a variable to be an *eigenvariable* of a rule. By this we mean that this variable is to be chosen in such a way that it does not occur in the conclusion of the rule. Note that this formal definition in all instances will turn out to be equivalent to the informal requirement that the variable occur only where it is "explicitly mentioned".

Definition 3.2.6 (Quantification Rules)**.** The rules of quantification in quantified propositional logic are the following rules.

$$\frac{\Gamma, A(\vec{a})}{\Gamma, \forall_k \vec{p} A(\vec{p})} \qquad \frac{\Gamma, A(\vec{\wp})}{\Gamma, \exists_k \vec{p} A(\vec{p})}$$

Here $\vec{a}$ have to be pairwise distinct eigenvariables. The $\vec{\wp}$ may be arbitrary propositional atoms. The formulae $\forall_k \vec{p} A(\vec{p})$ and $\exists_k \vec{p} A(\vec{p})$ are called the principal formulae of that inference.

Remark 3.2.7. It should be noted that in our introduction rules for the existential quantifier we only allow propositional atoms as witnesses. This differs from other presentations [39, 45, 18], where more complicated "target formulae" are allowed. The rationale is that our calculus is to follow AC^0-computations as closely as possible. Therefore, we consider the computation of the value of a variable as important in enough to keep track of its dependencies and hence have a separate rule for it. We think of the quantification rules are merely keeping track of the binding places of variables.

The fact that a variable (being a propositional atom!) can stand for a complicated value is reflected in the comprehension rule (Definition 3.2.11), that also can reflect independence of newly introduced variables, irrespectively of their later use as witnesses. See also Remark 3.2.12.

The importance of handling variable dependencies independent of binding places will become obvious when considering (in Subsection 4.3) the problem of evaluating a circuit. The intuitive reason, and an upper bound, is provided by the proof of Lemma 4.3.3. The corresponding lower bound is provided by Theorem 4.3.9.

Definition 3.2.8 (Cut Rule)**.** The *cut rule* of quantified propositional logic is the following rule.

$$\frac{\Gamma, A \qquad \Gamma, \neg A}{\Gamma}$$

The formula A in the cut rule is called the "cut formula".

We aim to design a calculus that appropriately reflects "AC^0-reasoning" and its sequential strength. One of the problems that can be solved in AC^0 is the following:

> Given truth values $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$, output q_i if i is the smallest index such that p_i is true.

A similar task in standard calculi of propositional logic would require a sequence of cuts, thus artificially increasing the height. As our investigations are essentially based on differences like constant versus logarithmic height, we cannot afford this increase. We therefore introduce a new rule allowing multiple cuts at once.

The fact that the multi-cut rule allows propositional induction to be shown by a constant height proof (Remark 3.2.10) will allow a constant height proof that evaluation of a certain (canonical) circuit implies the iteration principle, as can be seen from the proof of Corollary 4.3.8. The fact that this is a constant will be responsible for the very tight bound of proof heights and circuit heights established in Theorem 4.3.9. Another use of the multi-cut rule is the handling of the implicit induction principle due to the length function in the comprehension axiom. This can beautifully be seen in the proof of Lemma 6.2.7, where the propositional translation of the comprehension axiom is shown.

Definition 3.2.9 (Multi-Cut Rule)**.** The multi-cut rule is the rule

$$\frac{\ldots \qquad \Gamma, \Delta_i \qquad \ldots}{\Gamma}$$

where the Δ_i are sets of purely propositional formulae such that from the collection of the Δ_i the empty sequent can be derived by cuts only.

The width of the multi-cut rule is $\sum_i |\Delta_i|$, where $|\Delta_i|$ is the cardinality of the set Δ_i.

In other words, if from an arbitrary number of sequents, a sequent Γ can be derived by cuts on only purely propositional formulae, then this derivation of Γ counts as a *single* application of the multi-cut rule. For the calculus obtained to be a proof system in the sense of Cook and Reckhow [21] we require that the sequence of cuts be annotated in notations for proofs. However, as we are only interested in the number of rules applied we will never deal with notations for proofs.

Remark 3.2.10. The multi-cut rule allows one to prove purely propositional induction in constant depth.

In fact, from proofs of $\Gamma, \neg A_i, A_{i+1}$ for all $i < k$, we can conclude by a single inference that $\Gamma, \neg A_0, A_k$.

Next we will define the comprehension rule. It is motivated by the extension rule of extended Frege calculus [21, 36]. There, a new propositional variable may be introduced by the axiom $p \leftrightarrow \varphi$, if p is new, that is, does not occur anywhere earlier in the derivation. The extension rule says that if Γ can be derived from the assumption $\exists p(p \leftrightarrow \varphi)$, then it can also be derived without. Note that $\neg(\exists p(p \leftrightarrow \varphi)) \equiv \forall p \neg(p \leftrightarrow \varphi)$. As usual, the universal quantifier is expressed by the eigenvariable condition.

In order to take into account dependencies or independence of introduced variables we allow the introduction of several extension variables at the same time.

Definition 3.2.11 ($\mathcal{F}$-comprehension)**.** The *$\mathcal{F}$-comprehension rule of width k* is the rule

$$\frac{\Gamma, \neg(p_1 \leftrightarrow \varphi_1), \ldots, \neg(p_k \leftrightarrow \varphi_k)}{\Gamma}$$

where $\varphi_1, \ldots, \varphi_k \in \mathcal{F}$ and $p_1, \ldots, p_k$ are pairwise distinct eigenvariables that do not occur in any of the φ_i's (and by the eigenvariable condition they do not occur in Γ either).

The variables p_i are also called "extension variables" and the φ_i "extension formulae".

Remark 3.2.12. The name "$\mathcal{F}$-comprehension Rule" is justified by the fact, that it allows simple proofs of (propositional translations of) the comprehension axiom for formulae in $\mathcal{F}$. Consider the following derivation (where we

omit some side formulae; note that weakening is admissible).

$$\dfrac{\dfrac{\dfrac{\ldots \quad \overline{\overline{(p_i \leftrightarrow \varphi_i), \neg(p_i \leftrightarrow \varphi_i)} \quad \ldots}}{\bigwedge_k (p_i \leftrightarrow \varphi_i), \neg(p_1 \leftrightarrow \varphi_1), \ldots, \neg(p_k \leftrightarrow \varphi_k)}\bigwedge_k}{\exists_k \vec{p} \bigwedge_k (p_i \leftrightarrow \varphi_i), \neg(p_1 \leftrightarrow \varphi_1), \ldots, \neg(p_k \leftrightarrow \varphi_k)}\exists_k}{\exists_k \vec{p} \bigwedge_k (p_i \leftrightarrow \varphi_i)}\mathcal{F}\text{-comprehension}$$

It should be noted that the height of this derivation only depends on the φ_i and is independent of k. Proposition 3.3.11 will provide the needed proofs of the first sequents and will actually show that the heights depend only on the depths of φ_i's.

3.3 The AC^0-Tait Calculus

Definition 3.3.1 (AC^0-Tait)**.** The AC^0 *extended Frege calculus in Tait-style presentation*, or AC^0-Tait for short, is the calculus given by the following rules.

- The propositional rules (Definition 3.2.2).
- The parameter extensionality rule (Definition 3.2.4).
- The rules of quantification (Definition 3.2.6).
- The cut-rule (Definition 3.2.8) with cut-formulae restricted to purely propositional formulae.
- The multi-cut rule (Definition 3.2.9).
- The comprehension rule (Definition 3.2.11) for purely propositional formulae.

We assume all our proofs to be tree-like. This is not a restriction, as we only look at the height (not the size) of proofs.

Immediately by inspection of the rules, we note that weakening is admissible. This will be used tacitly in the sequel.

Remark 3.3.2. Obviously, the restricted cut-rule in Definition 3.3.1 of AC^0-Tait is subsumed by the multi-cut rule. However, we will later (in Definition 3.3.4) consider a variant of the AC^0-Tait calculus with a more liberal cut rule. To have a uniform transition between these calculi it seems natural to have the cut-rule always present and only change the set of formulae that may be cut on, despite the fact that it is superfluous in AC^0-Tait.

Definition 3.3.3. An AC^0-Tait proof is called w, c-*slim*, if all formulae occurring in the proof have size at most w, each multi-cut rule has width at most c, and each comprehension rule has at most c extension variables.

We write $\vdash^h_{w,c} \Gamma$ to denote that Γ has an AC^0-Tait proof of height at most h that is w, c-slim.

The calculus AC^0-Tait is our analogue to what in usual proof theoretic investigations corresponds to cut-free proofs. So we also consider a variant with proper cuts. As we shall see in Subsection 3.4, cuts can be eliminated at the usual price, that is, one exponentiation per quantifier alternation (Corollary 3.4.9). For this purpose we can treat $\exists_k$ and $\bigvee_k$ as belonging to the same "block of quantifiers" (compare Definition 3.1.22).

Definition 3.3.4 (AC^0-Tait with $\mathcal{C}$-Cuts)**.** If $\mathcal{C}$ is a set of formulae that contains all the purely propositional formulae and is closed under atomic substitutions we define the calculus "AC^0-Tait with $\mathcal{C}$-cuts" to be AC^0-Tait, but with the cut rule liberalised to formulae in $\mathcal{C}$.

We write $d \vdash^h_{\mathcal{C};w,c} \Gamma$ to denote that d is an AC^0-Tait with $\mathcal{C}$-cuts proof of Γ of height at most h that is w, c-slim. Note that this implicitly also implies that $\mathcal{C}$ is closed under atomic substitutions.

Remark 3.3.5. The restriction of $\mathcal{C}$ to be closed under atomic substitutions will become clear in Remark 3.3.7, which wouldn't hold otherwise (see also the proof of Lemma 3.3.6).

Lemma 3.3.6 (Atomic Substitution of Proofs)**.** *Assume that $\mathcal{C}$ is closed under atomic substitutions and let σ be an atomic substitution. If $\vdash^h_{\mathcal{C};w,c} \Gamma$ then $\vdash^h_{\mathcal{C};w,c} \Gamma\sigma$.*

Proof. Induction on the derivation. A $p, \bar{p}$-axiom might be replaced by a T-axiom. Note that by first renaming the eigenvariables (using the induction hypothesis for a different substitution) we can assume that the eigenvariables do not conflict with the range of σ. □

Remark 3.3.7. Lemma 3.3.6 in particular implies that we can always assume eigenvariables to be chosen sufficiently distinct from everything. This will be used tacitly in the sequel.

Lemma 3.3.8 (Weakening)**.** *If Δ is a set of formula, each of size at most w. If $\vdash^h_{\mathcal{C};w,c} \Gamma$ then $\vdash^h_{\mathcal{C};w,c} \Gamma, \Delta$.*

Proof. Induction on the derivation. Each rule can be reproduced identically; however compare Remark 3.3.7 which explains the need of $\mathcal{C}$ being closed under atomic substitutions—an assumption implicit in the definition of $\vdash^h_{\mathcal{C};w,c}$ Γ. □

Lemma 3.3.9 (Strengthening)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, \mathrm{F}$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma$.

Proof. Induction on the derivation. Each rule can be reproduced identically. Note that there is no rule with F as principal formula. □

Remark 3.3.10. Weakening and strengthening will be used tacitly in the sequel.

Proposition 3.3.11. *There is a* $C \in \mathbb{N}$ *such that* $\vdash^{C\cdot\mathsf{dp}(A)}_{\mathsf{sz}(A),0} A, \neg A$

Proof. Induction on A. The derivations

$$\frac{}{p, \bar{p}} \qquad \frac{}{\mathrm{T}, \mathrm{F}} \qquad \frac{}{\alpha_k(\wp_1, \ldots, \wp_k), \bar{\alpha}_k(\wp_1, \ldots, \wp_k)}$$

$$\frac{\ldots \quad \dfrac{A_i, \neg A_i}{\bigvee_k A_1 \ldots A_k, \neg A_i} \quad \ldots}{\bigvee_k A_1 \ldots A_k, \bigwedge_k (\neg A_1) \ldots (\neg A_k)} \qquad \frac{\dfrac{A(\vec{a}), \neg A(\vec{a})}{A(\vec{a}), \exists_k \vec{p}. \neg A(\vec{p})}}{\forall_k \vec{p}. A(\vec{p}), \exists_k \vec{p}. \neg A(p)}$$

are as desired. □

3.4 Cut-Elimination

Lemma 3.4.1 ($\bigwedge_k$-Inversion)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, \bigwedge_k \vec{A}$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma, A_i$.

Proof. Induction on the derivation. We can always use the induction hypothesis, except in the case of a $\bigwedge_k$-rule introducing $\bigwedge_k \vec{A}$. Here one of the premises is as desired. □

Corollary 3.4.2 ($\wedge$-Inversion)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, A \wedge B$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma, A$ *and* $\vdash^h_{\mathcal{C};w,c} \Gamma, B$.

Proof. This is the case $k = 2$ in Lemma 3.4.1. □

Lemma 3.4.3 ($\bigvee_k$-Inversion)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, \bigvee_k \vec{A}$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k$.

Proof. Induction on the derivation. We can always use the induction hypothesis, except in the case of a $\bigvee_k$-rule introducing $\bigvee_k \vec{A}$. Here the premise is as desired. □

Corollary 3.4.4 ($\vee$-Inversion)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, A \vee B$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma, A, B$.

Proof. This is the case $k = 2$ in Lemma 3.4.3. □

Lemma 3.4.5 ($\forall_k$-Inversion)**.** *If* $\vdash^h_{\mathcal{C};w,c} \Gamma, \forall_k \vec{p} A(\vec{p})$ *then* $\vdash^h_{\mathcal{C};w,c} \Gamma, A(\vec{\wp})$.

Proof. Every rule can be reproduced identically, except an $\forall_k$-rule introducing $\forall_k \vec{p} A(\vec{p})$. Note that by Remark 3.3.7 we may assume the $\vec{\wp}$ not to interfere with any eigenvariable condition.

In the case of a an $\forall_k$-rule concluding $\Gamma, \forall_k \vec{p} A(\vec{p})$ from $\Gamma, \forall_k \vec{p} A(\vec{p}), A(\vec{a})$ with eigenvariables $\vec{a}$ we first use the induction hypothesis to get $\Gamma, A(\vec{\wp}), A(\vec{a})$ and then use Lemma 3.3.6 to get $\Gamma, A(\vec{\wp}), A(\vec{\wp})$, noting that eigenvariable condition ensures that the substitution does not affect Γ or $A(\vec{\wp})$. □

Theorem 3.4.6. *Assume that* $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k$ *for some* $A_1, \ldots, A_k \in \Sigma\mathcal{C}$ *and assume that for all* $1 \leq i \leq k$ *we have* $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg A_i$. *Then* $\vdash^{h'+h+k}_{\mathcal{C};w,c} \Gamma$.

Proof. Induction on the first derivation, or, equivalently, induction on h'. We can reproduce all rules identically, except for those where the principal formula is among the $\vec{A}$. According to Definition 3.1.22 we distinguish the following cases.

- Case the last rule introduced a formula $A_i \in \mathcal{C}$, without loss of generality the formula A_k.

  So, assume that $\ell \in \mathbb{N}$ is a natural number and from derivations $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_{k-1}, A_k, \vec{A}^\circ_j$ for $1 \leq j \leq \ell$ it was concluded $\vdash^{h'+1}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_{k-1}, A_k$ by a rule with principal formula A_k, and premises $\vec{A}^\circ_j$ for $1 \leq j \leq \ell$. Here we allow the case $h' = 0$ if $\ell = 0$, that is, if the rule had no premises.

  Apply the induction hypothesis to all ℓ subderivations, considering A_k and $\vec{A}^\circ_j$ as part of the context(!), so that only $k-1$ formulae have to be removed. This yields derivations $\vdash^{h'+h+k-1}_{\mathcal{C};w,c} \Gamma, A_k, \vec{A}^\circ_j$ for $1 \leq j \leq \ell$. Now argue as follows.

$$(1 \leq j \leq \ell)\ \frac{\cdots \quad \vdash^{h'+h+k-1}_{\mathcal{C};w,c} \Gamma, A_k, \vec{A}^\circ_j \quad \cdots}{\mathrm{Cut}\ \dfrac{\vdash^{h'+h+k}_{\mathcal{C};w,c} \Gamma, A_k \qquad\qquad \vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg A_k}{\vdash^{h'+1+h+k}_{\mathcal{C};w,c} \Gamma}}$$

  Note that this derivation is as desired.

- Case the last rule introduced a formula $A_i \in \Sigma\mathcal{C} \setminus \mathcal{C}$ of the shape $\bigvee_\ell \vec{B}$, without loss of generality, assume it was the formula A_k.

  So assume that $A_k \equiv \bigvee_\ell \vec{B}$. Since $A_k \notin \mathcal{C}$ we may, by Definition 3.1.22 of the Σ-closure, conclude that $\vec{B} \in \Sigma\mathcal{C}$.

Moreover, assume that from $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k, B_{j_0}$ for some $1 \leq j_0 \leq \ell$ it was concluded $\vdash^{h'+1}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k$.

By the assumption of this theorem, we have a derivation $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg \bigvee_\ell \vec{B}$. Apply $\bigwedge_\ell$-inversion (Proposition 3.4.1) to this derivation to obtain an additional derivation $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg B_{j_0}$.

Now apply the induction hypothesis to $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k, B_{j_0}$ eliminating the $k+1$ formulae $A_1, \ldots, A_k, B_{j_0}$, obtaining $\vdash^{h'+1+k}_{\mathcal{C};w,c} \Gamma$ as desired.

- Case the last rule introduced a formula $A_i \in \Sigma\mathcal{C} \setminus \mathcal{C}$ of the shape $\exists_\ell \vec{p} A^\circ(\vec{p})$, without loss of generality assume that it was the formula A_k.

  By the Definition 3.1.22 of the Σ-closure we may conclude $A^\circ(\vec{p}) \in \Sigma\mathcal{C}$; see also Remark 3.1.23.

  So assume that from $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k, A^\circ(\vec{\wp})$ for some $\vec{\wp} \in \mathcal{A}$ it was concluded $\vdash^{h'+1}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k$.

  By the assumption of this theorem, we have a derivation $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \forall_\ell \vec{p} \neg A^\circ(\vec{p})$. Apply Lemma 3.4.5 to this derivation in order to obtain an additional derivation $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg A^\circ(\vec{\wp})$.

  Now apply the induction hypothesis to $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A_1, \ldots, A_k, A^\circ(\vec{\wp})$ eliminating the $k+1$ formulae $A_1, \ldots, A_k, A^\circ(\vec{\wp})$. This yields $\vdash^{h'+h+k+1}_{\mathcal{C};w,c} \Gamma$, which is as desired. □

Corollary 3.4.7. *Let $\mathcal{C}$ be closed under atomic substitutions, $A \in \Sigma\mathcal{C}$ and $h, h' \in \mathbb{N}$ natural numbers.*
If $\vdash^{h'}_{\mathcal{C};w,c} \Gamma, A$ and $\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg A$, then $\vdash^{h'+h+1}_{\mathcal{C};w,c} \Gamma$.

Proof. This is the special case $k = 1$ in Theorem 3.4.6. □

Lemma 3.4.8 (Cut Reduction)**.** *Assume that $\mathcal{C}$ is closed under atomic substitutions.*
If $\vdash^{h}_{\Sigma\mathcal{C};w,c} \Gamma$ then $\vdash^{2^h-1}_{\mathcal{C};w,c} \Gamma$.

Proof. First note that $2^h - 1 \geq h$ for all $h \geq 0$ and that $2^h - 1$ is strictly monotonic in h.

We prove the claim by induction on the derivation. We can reproduce every rule, except for a cut. So assume that $\vdash^{h+1}_{\Sigma\mathcal{C};w,c} \Gamma$ was concluded from $\vdash^{h}_{\Sigma\mathcal{C};w,c} \Gamma, A$ and $\vdash^{h}_{\Sigma\mathcal{C};w,c} \Gamma, \neg A$ by a cut. By our assumption on the cut-formulae we obtain $A \in \Sigma\mathcal{C}$.

Apply the induction hypothesis to both subderivations in order to obtain $\vdash^{2^h-1}_{\mathcal{C};w,c} \Gamma, A$ and $\vdash^{2^h-1}_{\mathcal{C};w,c} \Gamma, \neg A$. Now apply Corollary 3.4.7 to obtain $\vdash^{(2^h-1)+(2^h-1)+1}_{\mathcal{C};w,c} \Gamma$ which is as desired, since $(2^h-1)+(2^h-1)+1 = 2^h + 2^h - 1 = 2^{h+1} - 1$. □

Corollary 3.4.9. *Assume that $\mathcal{C}$ is closed under atomic substitutions.*
If $\vdash^h_{\Sigma\mathcal{C};w,c}\Gamma$ then $\vdash^{2^h}_{\mathcal{C};w,c}\Gamma$.

Proof. Immediately from Lemma 3.4.8 □

Knowing how to eliminate on level of quantifier alternation, we get the usual bounds for several quantifier alternations.

Proposition 3.4.10. *If $\vdash^h_{\Sigma^q_k(\alpha);w,c}\Gamma$ then $\vdash^{2_k(h)}_{\Sigma^q_0(\alpha);w,c}\Gamma$.*

Proof. Induction on k. If $k = 0$ the claim is trivial. For the inductive step we use Corollary 3.4.9 noting that $\vdash^h_{\Sigma^q_k(\alpha);w,c}\Gamma$ if and only if $\vdash^h_{\Pi^q_k(\alpha);w,c}\Gamma$ by the symmetry of cuts. □

4 The Sequential Iteration Principle

As mentioned in the introduction to Section 3, the calculus AC^0-Tait has a strong connection to the height of boolean circuits. This section will make this connection precise.

We will first (in Subsection 4.1) introduce iteration as an inherently sequential concept. This sequentiality is witnessed by the fact that circuits (of sub-exponential height) can only iterate as far as their height (Theorem 4.1.9). Formalising this principle in propositional logic (in Subsection 4.2) will yield a family of formulae that cannot have shallow proofs (Theorem 4.2.17). In fact, the lower bound on the height grows exponentially with the size of these formulae (Corollary 4.2.19). A minor variation (in Subsection 4.3) of these propositional formulae directly show the connection to circuit evaluation. Theorem 4.3.9 (together with Lemma 4.3.3) proves that, up to an additive constant, the height of a proof is the height of that very circuit of which it is proved that it can be evaluated.

Even though these results are highly encouraging, they heavily depend on the fact that we use *relativised* quantified propositional logic. To make these limits of the method more explicit we introduce (in Subsection 4.4) a calculus $\mathsf{AC}^\star$-Tait that extends AC^0-Tait in that it allows comprehension for *arbitrary* α-free formulae. So its strength is unlimited for unrelativised computations. Nevertheless, the same boundedness result holds (Theorem 4.4.5). We will later (in Subsection 6.4) use this calculus as target for translating $\mathsf{VNL}(\alpha)$ proofs into constant height propositional proofs. But already at this point it is instructive to see the limitations of the chosen method as well, besides all its potential and benefits.

4.1 Iteration as Sequential Principle

In this subsection we introduce our main measure of strength and show (in Lemma 4.1.12) that it can be used to separate the $\mathsf{AC}^k(\alpha)$-hierarchy. In later parts we will see how this measure can be applied in various ways to give meaningful results.

Our measure will very traditionally measure the amount of sequential time a model of computation can simulate. Parallel computation allows for reasonable complexity classes with sublinear time. As we shall see, our measure will be useful for the full range from constant to exponential growth.

The prototypical model for parallel computation are circuits and the notion of "time" here is the height of the circuits. So we use circuits as a point of reference to make sure our measure is well calibrated. We build on work by Takeuti [63] who developed a technique to separate some theories in weak

bounded arithmetic.

The idea is that computing the k'th iteration $f^k(0) = f(f(\ldots(f(0))))$ of a function f is essentially a sequential procedure, whereas shallow circuits represent parallel computation. So a circuit performing well in a sequential task has to be of big height. To avoid that the sequential character of the problem can be circumvented by precomputing all possible values, the domain of f is chosen big enough; we will consider functions $f\colon [2^n] \to [2^n]$.

Of course with such a big domain, we cannot represent such functions simply by a value table. That's how oracles come into play: oracles allow us to provide a predicate on strings as input, without the need of having an input bit for every string. In fact, the number of bits potentially accessible by an oracle gate is exponential in the number of its input wires.

Therefore we represent the i'th bit of $f(x)$ for $x \in \{0,1\}^n$ by whether or not the string $x\underline{i}$ belongs to the language of the oracle. Recall (Notation 2.1.41) that $\underline{i}$ is some canonical coding of the natural number i using $\log(n)$ bits.

Our argument can be summarized as follows. We assume a circuit of height h be given that supposedly computes the ℓ'th iterate of any function f given by the oracle. Then we construct, step by step, an oracle that fools this circuit, if $\ell > h$. To do so, for each layer of the circuit we decide how to answer the oracle questions, and we do this in a way that is consistent with the previous layers and such that all the circuit at layer i knows about f is at most the value of $f^i(0)$. Of course, to make this step-by-step construction possible we have to consider partial functions during our construction.

Definition 4.1.1 (ℓ-sequential)**.** A partial function $f\colon [2^n] \rightharpoonup [2^n]$ is called *ℓ-sequential* if for some $k \leq \ell$ it is the case that $0, f(0), f^2(0), \ldots, f^k(0)$ are all defined, but $f^k(0) \notin \mathsf{dom}(f)$.

Remark 4.1.2. In Definition 4.1.1 it is necessarily the case that $0, f(0), f^2(0), \ldots, f^k(0)$ are distinct.

Proof. If, say, $f^i(0) = f^{i+d}(0)$ for some $d > 0$, and $i + d \leq k$ then also $f^{k-d}(0) = f^k(0)$ and, in particular, $f^k(0) \in \mathsf{dom}(f)$, as $f^{k-d+1}(0)$ is defined. □

Example 4.1.3. The empty function is ℓ-sequential for any $\ell \in \mathbb{N}$. If f is a partial function with $f(0) = 0$ then f is *not* ℓ-sequential for any ℓ.

Lemma 4.1.4. *Let $n \in \mathbb{N}$ and $f\colon [2^n] \rightharpoonup [2^n]$ be an ℓ-sequential partial function. Moreover, let $M \subset [2^n]$ such that $|\mathsf{dom}(f) \cup M| < 2^n$. Then there is an $(\ell+1)$-sequential extension $f' \supseteq f$ with $\mathsf{dom}(f') = \mathsf{dom}(f) \cup M$.*

Proof. Let $a \in [2^n] \setminus (M \cup \mathsf{dom}(f))$. Such an a exists by our assumption on the cardinality of $M \cup \mathsf{dom}(f)$. Let f' be f extended by setting $f'(x) = a$ for all $x \in M \setminus \mathsf{dom}(f)$. This f' is as desired.

Indeed, assume that $0, f'(0), \ldots, f'^{\ell+1}(0), f'^{\ell+2}(0)$ are all defined. Then, since $a \notin \mathsf{dom}(f')$, all the $0, f'(0), \ldots, f'^{\ell+1}(0)$ have to be different from a. Hence these values have already been defined in f. But this contradicts the assumption that f was ℓ-sequential. □

Definition 4.1.5 (Bit Graph Function, Bit Graph Oracle)**.** To any natural number n and any partial function $f\colon [2^n] \rightharpoonup [2^n]$ we associate its *bit graph* $\alpha_{n,f}$ as a partial function $\alpha_{n,f}\colon \{0,1\}^{n+\log n} \rightharpoonup \{0,1\}$ in the obvious way. More precisely, $\alpha_{n,f}(uv)$ is the i'th bit of $f(x)$ if $f(x)$ is defined, and undefined otherwise, where u is a string of length n coding the natural number x and v is a string of length $\log n$ coding the natural number i.

If $f\colon [2^n] \to [2^n]$ is a total function, we define the *bit graph oracle* to be the set $\mathrm{A}_f = \{x \mid \alpha_{n,f}(x) = 1\} \subseteq \{0,1\}^{n+\log n}$.

Remark 4.1.6. Immediately from Definition 4.1.5 we note that f can be uniquely reconstructed from A_f.

Notation 4.1.7. If $A \subseteq \{0,1\}^*$ is an oracle, we denote by $A^{[n]} = \{x \in A \mid |x| = n + \log n\}$ the set of all strings in A of length $n + \log n$.

We now are mainly concerned with circuits with no inputs. For these circuits, the output only depends on the oracle. Requiring, however, that the output has a certain dependency on the oracle, the computational task becomes non-trivial. Before we (in Theorem 4.1.9) show a lower bound on the height of such a circuit solving a particular task, we make a simple observation which will show that the bound obtained is optimal.

Proposition 4.1.8. *Let n be a natural number. For every h there is a circuit C_h with no inputs, height $h+1$ and size $h \cdot n + 2$ such that for every oracle A the circuit C_h computes $f^h(0)$ for the (uniquely determined) $f\colon [2^n] \to [2^n]$ such that $\mathrm{A}_f = A^{[n]}$.*

Proof. At level 0 the circuit has two nodes that will represent the constants 0 and 1, i.e., we have an or-gate and an and-gate, both with no inputs.

At level i for $1 \leq i \leq h$, we have n oracle gates $c_{i,0}, \ldots, c_{i,n-1}$. All the oracle gates have $n+\log n$ inputs. The inputs $n \ldots n+\log n-1$ of $c_{i,j}$ code the numeral $\underline{j}$. Here the constant gates at level 0 are used. The inputs $0 \ldots n-1$ code the function computed at the lower level; that is, if $i > 1$, then they are the nodes $c_{i-1,0} \ldots c_{i-1,n-1}$ and if $i = 1$ we just use n times the constant 0 node.

It is easy to see that the circuit has the desired properties. □

Theorem 4.1.9. *Let C be any circuit with no inputs of depth h and size strictly less then 2^n. If C on oracle A computes correctly the last bit of $f^\ell(0)$ for the (uniquely determined) $f\colon [2^n] \to [2^n]$ such that $\mathrm{A}_f = A^{[n]}$, and if this is true for all oracles A, then $\ell \leq h$.*

Proof. Assume that such a circuit computes $f^\ell(0)$ correctly for all oracles. We have to find an oracle that witnesses $\ell \leq h$. First fix the oracle arbitrarily on all strings of length different from $n + \log n$. So, in effect we can assume that the circuit only uses oracle gates with $n + \log n$ inputs.

By induction on $k \geq 0$ we define partial functions $f_k\colon [2^n] \rightharpoonup [2^n]$ with the following properties. (Here we number the *levels* of the circuit $0, 1, \ldots, h-1$.)

- $f_0 \subseteq f_1 \subseteq f_2 \subseteq \ldots$
- The size $|\mathsf{dom}(f_k)|$ of the domain of f_k is at most the number of oracle gates with level strictly smaller than k.
- α_{n,f_k} determines the values of all oracle gates at levels strictly smaller than k.
- f_k is k-sequential.

We can take f_0 to be the totally undefined function, since $f^0(0) = 0$ by definition. As for the induction step let M be the set of all x of length n such that, for some $i < n$, the string $x\underline{i}$ is queried by an oracle gate at level k and let f_{k+1} be a k+1-sequential extension of f_k to domain $\mathsf{dom}(f_k) \cup M$ according to Lemma 4.1.4.

For $k = h$ we get the desired bound. As α_{n,f_h} already determines the values of all gates, the output of the circuit is already determined, but $f^{h+1}(0)$ is still undefined and we can define it in such a way that it differs from the output of the circuit. □

Remark 4.1.10. Inspecting the proof of Theorem 4.1.9 we note that it does not at all use what precisely the non-oracle gates compute, as long as the value only depends on the input, not on the oracle. In particular, the proof still holds if we consider subcircuits without oracle gates as a single complicated gate.

An immediate consequence of Theorem 4.1.9 is, that the $\mathsf{AC}^k(\alpha)$-classes form a strict hierarchy.

Definition 4.1.11 ($\mathcal{L}_g^A$)**.** If $g\colon \mathbb{N} \to \mathbb{N}$ is a function from the natural numbers to the natural numbers, and $A \subseteq \{0,1\}^*$ an oracle, we define the language

$$\mathcal{L}_g^A = \{x \mid \begin{array}{l}\text{the last bit of } f^{g(n)}(0) \text{ is 1,} \\ \text{where } n = |x| \text{ and } f \text{ is such that } A^{[n]} = \mathrm{A}_f\}\end{array}$$

We note that in Definition 4.1.11 the f is uniquely determined by A and the length of x.

Lemma 4.1.12. $\mathsf{AC}^k(\alpha) \neq \mathsf{AC}^{k+1}(\alpha)$

Proof. We applying Proposition 4.1.8 to $h = \log^{k+1}(n)$ and extend the obtained circuit $C_{\log^{k+1}(n)}$ to a circuit C'_n by adding n dummy input-nodes, and, at the same time, restrict the output list to its last element. Then $(C'_n)_{n\in\mathbb{N}}$ obviously is an $\mathsf{AC}^{k+1}(\alpha)$-circuit computing, parametrised by oracle A, the language $\mathcal{L}^A_{\log^{k+1}}$.

Now assume, for the sake of contradiction, that $(D_n)_{n\in\mathbb{N}}$ is an $\mathsf{AC}^k(\alpha)$-circuit computing $\mathcal{L}^A_{\log^{k+1}}$. Using that the D_n have polynomial, and hence in particular subexponential size, Theorem 4.1.9 now tells us, that, for large enough n, the circuit D_n has to have size $\log^{k+1}(n)$. But this contradicts our assumption, as $\log^{k+1} \notin \mathcal{O}(\log^k)$. □

For the uniform circuit classes, we can even show a stronger result: a single oracle is enough to separate them all.

Definition 4.1.13 ($\mathcal{U}$-uniform h, s-circuits)**.** Let $\mathcal{U}$ be a notion of uniformity, and $h, s\colon \mathbb{N} \to \mathbb{N}$ functions. The $\mathcal{U}$-uniform h, s-circuits are those circuit families $(C_n)_{n\in\mathbb{N}} \in \mathcal{U}$ of $\mathcal{U}$ such that C_n has depth at most $h(n)$ and size at most $s(n)$.

Theorem 4.1.14. *Let $\mathcal{U}$ be a notion of uniformity and h_c a family of functions such that for all $c \in \mathbb{N}$ the function h_{c+1} eventually strictly dominates h_c. Moreover, let s_c be a family of strictly subexponentially growing functions. Then there is a single oracle $A \subseteq \{0,1\}^*$ that simultaneously witnesses that $\mathcal{L}^A_{h_{c+1}}$ cannot be computed by $\mathcal{U}$-uniform h_c, s_c-circuits.*

Proof. Let $\mathcal{C}_0, \mathcal{C}_1, \ldots$ an enumeration of $\mathcal{U}$. Let (c_i, k_i) be an enumeration of all pairs of natural numbers.

We will construct natural numbers n_i, and sets A_i such that the following properties hold.

- The n_i strictly increase.
- A_i contains only strings of length $n_i + \log(n_i)$.
- If $\mathcal{C}_{k_i} = (C^{k_i}_n)_{n\in\mathbb{N}}$ and $C^{k_i}_{n_i}$ has depth at most $h_{c_i}(n_i)$ and size at most $s_{c_i}(n_i)$ then the language of $C^{k_i}_{n_i}$ with oracle $\bigcup_{j\leq i} A_j$ differs from $\mathcal{L}^{\bigcup_{j\leq i} A_j}_{h_{c_i+1}} \cap \{0,1\}^{n_i}$, and $C^{k_i}_{n_i}$ contains no oracle gates with $n_{i+1} + \log(n_{i+1})$ or more inputs.

Obviously $\bigcup_i A_i$ will be as desired.

At stage i take n_i big enough, so that it is bigger than all the previous n_j's and that $n_i + \log(n_i)$ is bigger than the maximal fan-in of all the oracle gates in all the circuits looked at so far; moreover take n_i big enough such that $s_{c_i}(n_i) < 2^{n_i}$ and $h_{c_i}(n_i) < h_{c_i+1}(n_i)$ which is possible as s_{c_i} has strictly subexponential growth and h_{c_i+1} dominates h_{c_i} eventually.

Look at the n_i'th circuit in the circuit family $\mathcal{C}_{k_i}$, and call it C. We may assume that C has height at most $h_{c_i}(n_i)$ and size at most $s_{c_i}(n_i)$ for otherwise there is nothing to show and we can choose A_i an arbitrary subset of $\{0,1\}^{n_i+\log(n_i)}$, say the empty set.

By Theorem 4.1.9 we find an $A_i \subset \{0,1\}^{n_i+\log(n_i)}$ such that C with oracle $\bigcup_{j\leq i} A_i$ does not solve the decision problem associated with $f^{h_{c_i+1}(n_i)}(0)$ for f given by $\mathrm{A}_f = A_i$. □

Corollary 4.1.15. *Let $\mathcal{U}$ be a notion of uniformity. Then there is a single oracle $A \subseteq \{0,1\}^*$ that witnesses the strictness of the $\mathcal{U}$-uniform $\mathsf{AC}^k(\alpha)$-hierarchy.*

4.2 Bounds in Propositional Logic

In Section 4.1 we have seen that iteration is a useful principle to separate circuits according to their height. We now go one step further and formalise this principle in propositional logic. This will give us lower bounds on the proof height in AC^0-Tait Calculus.

In this subsection we assume that n is big enough, so that $n+\log(n)$ and $2n$ are different. Note that this is the case if $n \geq 1$.

The intended meaning of $\alpha_{n+\log n}$ and α_{2n} is that they fix the values of a function $f\colon [2^n] \to [2^n]$ in the following way: $\alpha_{n+\log n}(i,x)$ is true iff the i'th bit of $f(x)$ is 1, and $\alpha_{2n}(i,x)$ is true iff $f^i(0) = x$, where $f^i(0)$ is the result of computing the i'th iteration of f on 0. Storing f by its bitgraph $\alpha_{n+\log n}$ automatically guarantees that a total function on $[2^n]$ is described, a property which would otherwise require adding more complex quantification to our principle.

Definition 4.2.1. We write "$f(p_1,\ldots,p_n) = q_1,\ldots,q_n$" for $\bigwedge_{i<n}(q_i \leftrightarrow \alpha_{\tilde{n}}(i,\vec{p}))$ where $\tilde{n} = n + \log(n)$. We write "$\vec{p} = \vec{q}$" for $\bigwedge_{i<n}(p_i \leftrightarrow q_i)$.

Definition 4.2.2. We write "$f^{p_1,\ldots,p_n}(0) = q_1,\ldots,q_n$" for $\alpha_{2n}(\vec{p},\vec{q})$.

It should be noted that "$f(0) = \vec{q}$" and "$f^1(0) = \vec{q}$" are not only different formulae, but are not even logically equivalent.

Definition 4.2.3. We write "$p_0, \ldots, p_{n-1} = q_0, \ldots, q_{n-1} + 1$" for the obvious AC^0-formulation of the successor relation, that is, for

$$\bigvee_i (\bigwedge_{j<i} p_j \wedge \neg p_i \wedge \bigwedge_{j<i} \neg q_j \wedge q_i \wedge \bigwedge_{j>i} (p_j \leftrightarrow q_j)) \; .$$

Our iteration principle will express that α_{2n} stores the graph of $i \mapsto f^i(0)$ for $i = 0, \ldots, \ell$. Here $\ell \leq n$ is a fixed number. Using the common idea that $\exists x. f^i(0) = x$ expresses that $f^i(0)$ can be computed, we can argue as follows. If $f^0(0)$ can be computed but $f^\ell(0)$ cannot, then there must be some i such that $f^i(0)$ can be computed but $f^{i+1}(0)$ cannot. The crux is now that this can be expressed using existential quantifiers only, which makes use of the trick that we are storing f by it's bit-graph. If $f^0(0) = 0$ and no m exists with $f^\ell(0) = m$, then there are m, m', i, i' with $i' = i + 1$ and $f^i(0) = m$ and $f(m) = m'$ and not $f^{i'}(0) = m'$. Prenexing this description and identifying the two independent occurrences of m gives us the following iteration formula and principle.

Definition 4.2.4. *The n, ℓ-iteration formula* $\Phi_{n,\ell}$ is the following purely propositional formula

$$\begin{aligned}
&\Phi_{n,\ell}(\vec{p}, \vec{p}\,', \vec{q}, \vec{q}\,') \;\equiv \\
&\quad \text{“}f^\ell(0) = \vec{p}\text{”} \vee \neg \text{“}f^0(0) = 0\text{”} \\
&\vee \quad (\text{“}\vec{q}\,' = \vec{q} + 1\text{”} \;\wedge\; \text{“}f^{\vec{q}}(0) = \vec{p}\text{”} \;\wedge\; \text{“}f(\vec{p}) = \vec{p}\,'\text{”} \;\wedge\; \neg \text{“}f^{\vec{q}\,'}(0) = \vec{p}\,'\text{”})
\end{aligned}$$

The *n, ℓ-iteration principle* is the formula

$$\exists_{4n} \vec{p}\vec{p}\,'\vec{q}\vec{q}\,' .\; \Phi_{n,\ell}(\vec{p}, \vec{p}\,', \vec{q}, \vec{q}\,') \; .$$

Definition 4.2.5. A *partial propositional assignment* is a finite partial mapping from the propositional variables to $\{\mathrm{T}, \mathrm{F}\}$.

A *partial parameter assignment* is any partial mapping (not necessarily finite) from atomic parameters $\alpha_k(\vec{\wp})$, with $\wp_i \in \{\mathrm{T}, \mathrm{F}\}$, to $\{\mathrm{T}, \mathrm{F}\}$.

Notation 4.2.6. In the context of propositional logic, we use "valuation" as another word for partial (propositional or parameter) assignment. We use η to range over valuations. In accordance with set theoretic notions we write the empty valuation as $\emptyset$.

Definition 4.2.7 ($A\eta$)**.** If A is a quantified propositional formula and η a partial propositional assignment, we define $A\eta$ by induction on A. For p a propositional variable with $p \in \mathsf{dom}(\eta)$ we set $p\eta \equiv \eta(p)$ and $\bar{p}\eta \equiv \neg\eta(p)$. For $p \notin \mathsf{dom}(\eta)$ we set $p\eta \equiv p$ and $\bar{p}\eta \equiv \bar{p}$. The remaining cases are defined

homomorphically, e.g., $(\bigwedge_k \vec{A})\eta \equiv \bigwedge_k \vec{A\eta}$. In particular $\alpha_k(\wp_1, \ldots, \wp_k)\eta \equiv \alpha_k(\wp_1\eta, \ldots, \wp_k\eta)$.

If A is a closed purely propositional formula and η a partial parameter assignment, we define $A\eta$ by induction on A. For $\alpha_k(\vec{\wp})$ with $\alpha_k(\vec{\wp}) \in \mathsf{dom}(\eta)$ we set $(\alpha_k\vec{\wp})\eta \equiv \eta(\alpha_k(\vec{\wp}))$ and $(\bar{\alpha}_k\vec{\wp})\eta \equiv \neg\eta(\alpha_k(\vec{\wp}))$. Otherwise we set $(\alpha_k\vec{\wp})\eta \equiv \alpha_k(\vec{\wp})$ and $(\bar{\alpha}_k\vec{\wp})\eta \equiv \bar{\alpha}_k(\vec{\wp})$. The remaining cases are defined homomorphically.

Notation 4.2.8. If $\Gamma = \{A_1, \ldots, A_k\}$ is a set of formulae we write $\Gamma\eta$ for $\{A_1\eta, \ldots, A_k\eta\}$.

By a trivial induction on A we get

Proposition 4.2.9. $\neg(A\eta) \equiv (\neg A)\eta$

Note the difference between a partial assignment, where the range is in $\{\mathrm{T}, \mathrm{F}\}$ and a substitution, where the range can be more liberal. In particular, the range of a partial assignment cannot be moved any more. This is made precise in the following lemma that will be used tacitly in the sequel.

Lemma 4.2.10. *If $\eta \subset \eta'$ are partial propositional assignments and A is a quantified propositional formula such that $A\eta$ is closed, then $A\eta \equiv A\eta'$.*

If $\eta \subset \eta'$ are partial parameter assignments and A is a closed purely propositional formula such that $A\eta$ is α-free, then $A\eta \equiv A\eta'$.

Proof. Trivial induction on A. □

Definitions 4.1.1 and 4.2.11 encode the crucial idea of our proof of the boundedness Theorem 4.2.17. Eventually we will be working upwards through a single path of a given proof, and partially define a function $f\colon [2^n] \rightharpoonup [2^n]$ in order to falsify all quantifier free formulae on this path. We want to do this in such a way, that, at level h, only $0, f(0), \ldots, f^{h-1}(0)$ are defined. But, to assign a truth value to a quantifier free formula, we not only have to set the parameter bits that encode the relation "$f(x) = y$", but also those that encode the iterations of f of the form "$f^k(0) = y$".

The idea is to assign them values consistent with what we have so far and also consistent with our strategy on how we plan to extend f. As we want to keep $f^h(0)$ undefined, all the values in $\mathsf{dom}(f)$ are "forbidden" anyway for the next extension of f. Note that, if $f^i(0)$ is defined and $f^i(0) = f^j(0)$ for some $i < j$, then all the values $f^k(0)$ are already defined.

Definition 4.2.11 (η_f)**.** If $n \in \mathbb{N}$ is a natural number and $f\colon [2^n] \rightharpoonup [2^n]$ a partial function, we associate with f, or actually the pair n, f, a partial parameter assignment η_f as follows.

For $j \in [n]$, $x \in [2^n]$ with $f(x)$ defined, say $f(x) = \langle \vec{r} \rangle \in [2^n]$, we set $\eta_f(\alpha_{n+\log(n)}(j, x)) = r_j$. Otherwise $\eta_f(\alpha_{n+\log(n)}(j, x))$ is undefined.

For $x, \ell \in [2^n]$ we set $\alpha_{2n}(\ell, x) = \mathrm{T}$ if $f^\ell(0)$ is defined and equal to x; otherwise we set $\alpha_{2n}(\ell, x) = \mathrm{F}$ if $x \in \mathsf{dom}(f)$; otherwise $\alpha_{2n}(\ell, x)$ is undefined.

For $k \notin \{2n, n + \log(n)\}$ we set $\eta_f(\alpha_k(\vec{\wp}))$ arbitrarily, say F. Also, if $\vec{p} \in \{\mathrm{T}, \mathrm{F}\}^{\log n} \setminus [n]$, we set $\alpha_{n+\log n}(\vec{p}, \vec{q})$ arbitrarily, say F.

"Good extensions" of partial functions are those that comply with the above idea, that is, those that do not assign new values that are already in the domain.

Definition 4.2.12. If $f, f' \colon [2^n] \rightharpoonup [2^n]$ are partial functions, and $f \subset f'$ then f' is called a *good extension* of f, if $\forall x \in \mathsf{dom}(f')(x \in \mathsf{dom}(f) \vee f'(x) \notin \mathsf{dom}(f))$.

Remark 4.2.13. If $f \subset f'$ and $f' \subset f''$ are good extensions, then so is $f \subset f''$.

Proof. Let $x \in \mathsf{dom}(f'')$ and assume $x \notin \mathsf{dom}(f)$. We have to show that $f''(x) \notin \mathsf{dom}(f)$. If $x \in \mathsf{dom}(f')$ then $f''(x) = f'(x) \notin \mathsf{dom}(f)$, since $f \subset f'$ is a good extension; otherwise $x \notin \mathsf{dom}(f')$, so $f''(x) \notin \mathsf{dom}(f') \supset \mathsf{dom}(f)$, since $f' \subset f''$ is a good extension. □

Proposition 4.2.14. *If $f \subset f'$ is a good extension, then $\eta_f \subset \eta_{f'}$.*

Proof. By inspecting Definition 4.2.11 we note that the only case we have to exclude is, that $\eta_f(\alpha_{2n}(\ell, x)) = \mathrm{F}$ and $\eta_{f'}(\alpha_{2n}(\ell, x)) = \mathrm{T}$.

This might happen, if $x \in \mathsf{dom}(f)$ and $f^\ell(0)$ is undefined, but $f'^\ell(0) = x$. Let k be maximal such that $f^k(0)$ is defined. Clearly $0 \leq k < \ell$. By induction on i we show that for $i < \ell$ we have $f'^{(\ell-i)}(0) \in \mathsf{dom}(f)$. For $i = \ell - k$ this gives the desired contradiction, since $\mathsf{dom}(f) \ni f'^{(\ell-(\ell-k))}(0) = f'^k(0) = f^k(0)$ would imply that $f^{k+1}(0)$ is defined.

For $i = 0$ nothing is to show, since $f'^\ell(0) = x \in \mathsf{dom}(f)$. So let $0 < i < \ell$. By induction hypothesis we know $\mathsf{dom}(f) \ni f'^{\ell-(i-1)}(0) = f'(f'^{\ell-i}(0))$ and, since $f \subset f'$ is good, we conclude that $f'^{\ell-i}(0)$ was already in the domain of f. □

Inspecting the proof of Lemma 4.1.4 we note that the partial function f' constructed there is not only an extension of f, but indeed a good one. Recall that the value a chosen to be assigned to all points that have to enter the domain is taken from outside the domain.

This observation is formalised as

Lemma 4.2.15. *Let $n \in \mathbb{N}$ and $f\colon [2^n] \rightharpoonup [2^n]$ be an ℓ-sequential partial function. Moreover, let $M \subset [2^n]$ such that $|\mathsf{dom}(f) \cup M| < 2^n$. Then there is an $(\ell+1)$-sequential good extension f' of f with $\mathsf{dom}(f') = \mathsf{dom}(f) \cup M$.*

Lemma 4.2.16. *For every closed, purely propositional, formula A of size ℓ there is a set $M \subset [2^n]$ such that $|M| \leq \ell$ and for every function f with $M \subset \mathsf{dom}(f)$ it holds that $A\eta_f$ is α-free.*

Proof. Let M be the set of all $x \in [2^n]$ such that an atom of the form $\alpha_{n+\log(n)}(j,x)$ or $\alpha_{2n}(k,x)$ occurs in A.

Note that $x \in \mathsf{dom}(f)$ forces $\eta_f(\alpha_{2n}(k,x))$ to have a definite value (F unless $f^k(0) = x$, in which case it would be T). □

Theorem 4.2.17. *Let k, n, w, c be natural numbers with $c \cdot w \geq 2$. Assume $\vdash^h_{w,c} \Gamma$ with $\Gamma = \Delta, \exists_{4n}\vec{r}\Phi_{n,\ell}(\vec{r})$, where $\Phi_{n,\ell}$ is the n,ℓ-iteration formula. Let η be a partial propositional assignment and $f\colon [2^n] \rightharpoonup [2^n]$ be k-sequential. Assume $|\mathsf{dom}(f)| + cwh < 2^n$. If each element of Δ is purely propositional, and $\Delta\eta\eta_f$ closed, α-free, and false then $\ell \leq k + h$.*

Proof. We argue by induction on h with case distinction according to the last rule of the proof.

The last rule cannot be a propositional axiom, as axioms cannot have $\exists_{4n}\vec{r}\Phi_{n,\ell}(\vec{r})$ as a principal formula; however, all the formulae in $\Delta\eta\eta_f$ are false so Δ cannot be a tautology, as it would have to be, as the calculus is sound. In the case of an $\bigvee_k$-inference apply the induction hypothesis, in the case of an $\bigwedge_k$-inference, the induction hypothesis is applicable to at least one of the subderivations. The last rule cannot be an $\forall_j$-rule as this would require a quantified formula in Δ.

If the last rule is a multi-cut rule

$$\frac{\ldots \Gamma, \Delta_i \ \ldots}{\Gamma}$$

we know, since the proof is w,c-slim, that $\bigcup_i \Delta_i$ contains at most c formulae of size at most w. Let $\eta' \supset \eta$ such that all $\Delta_i\eta'$ are closed. Let M be the union of the sets asserted by Lemma 4.2.16 for the formulae in $\bigcup_i \Delta_i\eta'$. Then $|M| \leq c \cdot w$. We extend f in a good way to some $(k+1)$-sequential f' with $\mathsf{dom}(f') = \mathsf{dom}(f) \cup M$. Noting that all the $\Delta_i\eta'\eta_{f'}$ are sets of α-free, closed, purely propositional formulae we can assign them truth values. Since, by cuts we can derive the empty sequent from the sets Δ_i, and hence also from the sets $\Delta_i\eta'\eta_{f'}$, one of them has to contain only false formulae. Apply the induction hypothesis to this subderivation.

The case of a cut rule is similar, but easier.

Assume that the last rule was a parameter extensionality rule as follows.

$$\frac{\Gamma, \alpha_j(\wp_1, \ldots, \wp_j) \qquad \ldots \Gamma, \wp_i \leftrightarrow \wp_i' \ldots \qquad (1 \leq i \leq j)}{\Gamma, \alpha_j(\wp_1', \ldots, \wp_j')}$$

Extend η to some η' assigning values to all the $\vec{\wp}$. If for some $1 \leq i \leq j$ we have $\wp\eta' \neq \wp'\eta'$ we can apply the induction hypothesis to the corresponding subderivation. Otherwise $(\alpha_k(\vec{\wp}))\eta'\eta_f \equiv (\alpha_k(\vec{\wp}'))\eta'\eta_f$ and we can apply the induction hypothesis to the first subderivation.

Assume that the last inference rule was an $\exists_j$-rule.

$$\frac{\Gamma, \Phi_{n,\ell}(\vec{\wp}, \vec{\wp}', \vec{\wp}'', \vec{\wp}''')}{\Gamma}\, \exists_{4n}$$

We can extend η to η' such that there are natural numbers m, m', i, i' such that $\vec{\wp}\eta' = m$, $\vec{\wp}'\eta' = m'$, $\vec{\wp}''\eta' = i$ and $\vec{\wp}'''\eta' = i'$. If $\ell \leq k$ there is nothing to show. Otherwise, we will argue as follows that $\Phi_{n,\ell}(m, m', i, i')\eta_{f'}$ can be falsified by choosing an appropriate $(k{+}1)$-sequential good extension f' of f. Since $\ell > k$, for every good $(k{+}1)$-sequential extension f' of f we have $f'^{(\ell+1)}(0)$ undefined. Hence for any such f' with $m \in \mathsf{dom}(f')$ we know that $f'^{(\ell)}$ is either undefined or different from m (for otherwise $f'^{(\ell+1)}(0)$ would be defined). In either case $\eta_{f'}(\alpha_{2n}(\ell, m)) = \mathrm{F}$. Recall that adding a value m to the domain of f' ensures that $\eta_{f'}(\alpha_{2n}(\ell, m))$ has a definite value. The second disjunct $\neg$"$f^0(0) = 0$" is falsified by $\eta_{f'}$ for any f'. For the last disjunct "$i' = i{+}1$" $\wedge$ "$f^i(0) = m$" $\wedge$ "$f(m) = m'$" $\wedge \neg$"$f^{i'}(0) = m'$", we may assume that $i' = i + 1$, for otherwise it is falsified anyway. For any f' with $m, m' \in \mathsf{dom}(f')$ we know that $\eta_{f'}$ assigns definite truth values to "$f^i(0) = m$", "$f(m) = m'$", and "$f^{i'}(0) = m'$". If the first two conjuncts are assigned T, than this can only be if $f'^i(0) = m$ and $f'(m) = m'$. But in this case $f'^{i+1}(0) = m'$, so $\neg$"$f^{i+1}(0) = m'$" is assigned F. Altogether we can take any $(k{+}1)$-sequential good extension f' of f with $\mathsf{dom}(f') = \mathsf{dom}(f) \cup \{m, m'\}$. Then $\Phi_{n,\ell}(\vec{\wp}, \ldots)\eta'\eta_{f'}$ is α-free, closed, purely propositional and false and we can apply the induction hypothesis (recalling that we assumed $wc \geq 2$).

The last remaining case is that the last rule was a comprehension rule

$$\frac{\Gamma, \neg(p_1 \leftrightarrow \varphi_1), \ldots, \neg(p_j \leftrightarrow \varphi_j)}{\Gamma}$$

where the φ_i are purely propositional, the $\vec{p}$ are eigenvariables, and, since the proof is w, c-slim, $j \leq c$. Let $\eta'' \supset \eta$ be such that all $\varphi_i\eta''$ are closed. Let M_i be the set asserted by Lemma 4.2.16 for $\varphi_i\eta''$. Extend in a good way f to a $(k{+}1)$-sequential f' with $\mathsf{dom}(f') = \mathsf{dom}(f) \cup \bigcup_i M_i$. Due to the eigenvariable condition we can assume without loss of generality that $\vec{p} \notin \mathsf{dom}(\eta'')$. Extend

η'' to η' by setting p_i to the truth value of $\varphi_i \eta'' \eta_{f'}$. We then can apply the induction hypothesis.

This finishes the proof. □

Corollary 4.2.18. *If* $\vdash^h_{w,c} \exists_{4n} \vec{r} \Phi_{n,\ell}(\vec{r})$ *and* $cwh < 2^n$ *for some* c, w *with* $cw \geq 2$ *then* $h \geq \ell$.

Proof. The special case $\Delta = \emptyset$, $\eta = \emptyset$, $f = \emptyset$, and $k = 0$ in Theorem 4.2.17. □

Corollary 4.2.19. *There is a family of polynomial size* $\Sigma^q_1(\alpha)$*-formula, such that every* AC^0*-Tait proof with polynomially branching rules and polynomial size formulae requires exponential height.*

Proof. As Corollary 4.2.18 shows, the family $(\exists_{4n} \vec{r} \Phi_{n,2^n-1}(\vec{r}))_{n \in \mathbb{N}}$ is as desired. It should be noted that these formuale indeed only grow polynomially, as, of course, the number $2^n - 1$ can be represented by n bits. □

4.3 Circuit Evaluation

This subsection shows an application of Subsection 4.2. The height of a proof showing that a circuit can be evaluated is, up to an additive constant, the height of that very circuit.

Definition 4.3.1 (Circuit-Evaluation Formula)**.** Let C be a circuit with empty input list and nodes $n_1, \ldots, n_k$. Then we define the *evaluation formula associated with* C as the formula $\Psi_C(\vec{p})$ where $p_1, \ldots, p_k$ are propositional variables associated with nodes $n_1, \ldots, n_k$, respectively. Ψ_C is the conjunction of the conditions for each node. If the node i is an and-gate, then the associated condition is

$$p_i \leftrightarrow \bigwedge_\ell p_{i_1} \ldots p_{i_\ell}$$

where $n_{i_1}, \ldots, n_{i_\ell}$ are the inputs for node i; the conditions for or-gates and not-gates are similar. In the special cases of an "and" or or–gate without inputs, we use the constants T and F, respectively.

For an oracle gate, the condition is

$$p_i \leftrightarrow \alpha_\ell(p_{i_1}, \ldots, p_{i_\ell})$$

where $\alpha n_{i_1}, \ldots, n_{i_\ell}$ is the labelling of that node. Similarly for a negated oracle gate.

Remark 4.3.2. It should be noted that Ψ_C is a formula of *constant* depth, irrespective of the shape of the circuit. However, as we shall see, the height of the proof needed to prove that this circuit can be evaluated depends on the actual structure of the circuit.

Lemma 4.3.3. *If C is a circuit of height h, then there is a proof of height $h + \mathcal{O}(1)$ for $\exists_k \vec{p}\, \Psi_C(\vec{p})$.*

Proof. For $0 \leq \ell < h$ let $p_{i_1}^{(\ell)}, \ldots, p_{i_{k_\ell}}^{(\ell)}$ be the variables associated with the nodes of level ℓ. So a variable $p_i^{(\ell)}$ depends only on variables $p_j^{(\ell')}$ for some $\ell' < \ell$. We write $C_i^{(\ell)}$ for the condition associated with $p_i^{(\ell)}$. Then the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdots}}{\exists_k \vec{p}\, \Psi_C, \neg(\vec{p}^{(h)} \leftrightarrow \vec{C}^{(h)})}\,(\text{comp})}{\exists_k \vec{p}\, \Psi_C}\,(\text{comp})}}{\exists_k \vec{p}\, \Psi_C, \neg(\vec{p}^{(h)} \leftrightarrow \vec{C}^{(h)}), \neg(\vec{p}^{(h-1)} \leftrightarrow \vec{C}^{(h-1)})}\,(\text{comp})}}{\exists_k \vec{p}\, \Psi_C, \neg(\vec{p}^{(h)} \leftrightarrow \vec{C}^{(h)}), \ldots, \neg(\vec{p}^{(2)} \leftrightarrow \vec{C}^{(2)}), \neg(\vec{p}^{(1)} \leftrightarrow \vec{C}^{(1)})}\,(\text{comp})}}{\Psi_C, \neg(\vec{p}^{(h)} \leftrightarrow \vec{C}^{(h)}), \ldots, \neg(\vec{p}^{(2)} \leftrightarrow \vec{C}^{(2)}), \neg(\vec{p}^{(1)} \leftrightarrow \vec{C}^{(1)})}\,(\exists_k)}}{\dots \quad p_i^{(\ell)} \leftrightarrow C_i^{(\ell)}, \neg(p_{i_j}^{(\ell)} \leftrightarrow C_{i_j}^{(\ell)}) \quad \dots \quad (0 \leq \ell < h),\ (1 \leq j < k_\ell)}\,\bigwedge$$

(Proposition 3.3.11)

is as desired. □

We now show the following lower bound. Consider a proof that a circuit can be evaluated. If the circuit has height h, then the proof has to have height at least $h - \mathcal{O}(1)$. As we know from Proposition 4.1.8, a circuit of height $h+1$ can compute the h'th iterate of the function given by α. From the fact that this circuit can be evaluated, we can conclude that the h-iteration principle holds. In the following let C_h be the circuit given by Proposition 4.1.8. We also assume the size parameter n to be understood; we set $\tilde{n} = n + \log n$. Immediately from the definition we get

Proposition 4.3.4. *$\Psi_{C_h}(\vec{w})$ is the conjunction of the clauses $w_0' \leftrightarrow \mathrm{F}$, $w_1' \leftrightarrow \mathrm{T}$ and the conjuncts "$f(\vec{w}^{(\ell)}) = \vec{w}^{(\ell+1)}$". The latter is built of the formulae $w_j^{(\ell+1)} \leftrightarrow \alpha_{\tilde{n}}(\vec{w}^{(\ell)}, \vec{u}^{(j)})$ of $0 \leq \ell < h-1$ and $0 \leq j < n$, where we used the abbreviations $w_i^{(0)} \equiv w_0'$, and $\vec{u}^{(j)}$ for a list of w_0' and w_1' coding the (binary) numeral j in the obvious way.*

Proof. Immediate by the Definition of the evaluation formula 4.3.1 and the definition of the circuit C_h in Proposition 4.1.8. □

As an immediate consequence we obtain

Proposition 4.3.5. *Ψ_{C_h} is purely propositional. Moreover there is a constant c such that for all $h, n \geq 1$ we have $\mathsf{sz}(\Psi_{C_h}) \leq c \cdot n \cdot h$ and $\mathsf{dp}(\Psi_{C_h}) \leq c$.*

Proof. Immediate from Proposition 4.3.4. □

Proposition 4.3.6. *For some constant c, if $i \in [n-1]$ for some $n \geq 2$, $\vec{p} = i$, and $\vec{q} = i + 1$ then $\vdash^c_{c \cdot \log n, 1}$ "$\vec{q} = \vec{p} + 1$".*

Proof. Immediately from Definition 4.2.3 of the successor relation and propositional logic in AC^0-Tait. □

In the following, for $1 \leq \ell < n$, we set $\Delta_\ell \equiv \neg$"$f^{\ell-1}(0) = \vec{w}^{(\ell-1)}$", "$f^\ell(0) = \vec{w}^{(\ell)}$". Recall that "$f^{\vec{p}}(0) = \vec{q}$" is a shorthand for $\alpha_{2n}(\vec{p}, \vec{q})$. So, by resolution the Δ_ℓ imply $\neg$"$f^0(0) = \vec{w}^{(0)}$", "$f^h(0) = \vec{w}^{(h)}$".

Lemma 4.3.7. *For some constant c it holds for all $n, h \geq 1$ that $\vdash^c_{cnh,1}$ $\Delta_\ell, \exists_{4n}\vec{u}\Phi_{n,h}(\vec{u}), \forall_{h \cdot n+2}\vec{w}\neg\Psi_{C_h}(\vec{w})$ where $\Phi_{n,h}(\vec{u})$ the n, h-iteration formula, as defined in Definition 4.2.4.*

Proof. First note, that there are constant height proofs of the following sequents.

- "$f(\vec{w}^{(\ell-1)}) = \vec{w}^{(\ell)}$", $\neg\Psi_{C_h}(\vec{w})$
- "$f^{\ell-1}(0) = \vec{w}^{(\ell-1)}$", $\neg$"$f^{\ell-1}(0) = \vec{w}^{(\ell-1)}$"
- "$f^\ell(0) = \vec{w}^{(\ell)}$", $\neg$"$f^\ell(0) = \vec{w}^{(\ell)}$"
- "$(\ell + 1) = \ell + 1$"

Therefore applications of an $\bigwedge_4$-rule followed by an $\vee$-rule and an $\exists_{4n}$-rule gives us $\exists_{4n}\vec{u}\Phi_{n,h}(\vec{u}), \neg\Psi_{C_h}(\vec{w}), \Delta_\ell$ from where we get the desired derivation by an $\forall_{nh+2}$-rule. □

Corollary 4.3.8. *For some c we have that $\vdash^c_{cnh,ch}$ $\exists_{4n}\vec{u}\Phi_{n,h}(\vec{u}), \forall_{h \cdot n+2}\vec{w}\neg\Psi_{C_h}$ holds for all $n, h \geq 1$.*

Proof. Apply a multi-cut rule to the derivations of Lemma 4.3.7 to obtain $\neg$"$f^0(0) = \vec{w}^{(0)}$", "$f^h(0) = \vec{w}^{(h)}$", $\exists_{4n}\vec{u}\Phi_{n,h}(\vec{u}), \forall_{h \cdot n}\vec{w}\neg\Psi_{C_h}(\vec{w})$. Two $\vee$-rules and an $\exists_{4n}$-rule finish the proof. □

Theorem 4.3.9. *There are natural numbers c, C such that for all sufficiently large n, h whenever $c^2 \cdot h^2 n < 2^n$ and $\vdash^{h'}_{c \cdot nh, ch} \exists_{nh+2} \vec{w} \Psi_{C_h}(\vec{w})$ then $h' \geq h - C$.*

Proof. Assume $\vdash^{h'}_{\tilde{w},\tilde{c}} \exists_{nh+2} \vec{w} \Psi_{C_h}$. By Corollary 4.3.8 we have (for sufficiently large n) a derivation $\vdash^{c_2}_{c_1 nh, c_1 h} \exists_{4n} \vec{u} \Phi_{n,h}(\vec{u}), \forall_{hn} \vec{w} \neg \Psi_{C_h}$. Therefore, by Corollary 3.4.7, we get $\vdash^{h'+c_2+1}_{\max\{w, c_1 nh\}, \max\{\tilde{c}, c_1 h\}} \exists_{4n} \vec{u} \Phi_{n,h}(\vec{u})$. So, by Corollary 4.2.18, we get $h' + c_2 + 1 \geq h$, provided $\max\{\tilde{w}, c_1 nh\} \cdot \max\{\tilde{c}, c_1 h\} < 2^n$. □

An immediate consequence of Theorem 4.3.9 is that a proof of Ψ_{C_h} requires height $h - \mathcal{O}(1)$, for all h growing sub-exponentially with n.

4.4 Limits of the Method: Unrelativised Computation

Recalling the proof of Theorem 4.2.17, one notes that the only reason for restricting the comprehension formulae was to avoid fixing too many values of α and hence the partially constructed function f; Lemma 4.2.16 was an essential tool.

However, a trivial way of avoiding the need of fixing too many values of α is considering α-free formulae. Despite the simple-mindedness of this approach it will turn out useful to also add comprehension for these kind of formulae to our propositional calculus. The resulting calculus, called $\mathsf{AC}^\star$-Tait, will be used in Subsection 6.4 as a target calculus for the propositional translation of $\mathsf{VNL}(\alpha)$. This embedding will, in Subsection 7.3, be used to obtain matching upper and lower bounds for strength of that theory.

We will now formally introduce the calculus $\mathsf{AC}^\star$-Tait and show that Theorem 4.2.17 extends to this calculus as well. This result, while being useful for obtaining precise strength measures, also shows the limits of our method: while being very sensitive to nesting depth of oracle queries (up to a constant, as we have seen in Theorem 4.3.9 and Lemma 4.3.3) it is blind to the cost of unrelativised computation.

Definition 4.4.1 ($\mathsf{AC}^\star$-Tait)**.** The calculus $\mathsf{AC}^\star$-Tait is defined to be AC^0-Tait (Definition 3.3.1) extended by comprehension (Definition 3.2.11) for arbitrary α-free formulae.

So far, we have only defined (in Definition 4.2.7) how we apply partial parameter assignments to purely propositional formulae; however, it is obvious how to apply a partial parameter assignment to a parameter free formula: just leave it, as it is.

Definition 4.4.2. If A is a α-free and η a partial parameter assignment, then $A\eta$ is defined to be A.

Lemma 4.4.3 (Atomic Substitution for $\mathsf{AC}^\star$-Tait)**.** *If $\mathsf{AC}^\star$-Tait proves Γ with a w, c-slim proof of height h and if σ is an atomic substitution then $\mathsf{AC}^\star$-Tait also proves $\Gamma\sigma$ with a w, c-slim proof of height h.*

Proof. Induction on h and following Lemma 3.3.6. Note that the property of being α-free is closed under (atomic) substitutions. □

Following Remark 3.3.7 we will, for $\mathsf{AC}^\star$-Tait as well, tacitly assume eigenvariables to be chosen sufficiently distinct where this is useful.

Inspection of the proof of Theorem 3.4.6 reveals that cut-elimination is not affected by any rule that does not have a principal formula. In particular, as the only difference between AC^0-Tait and $\mathsf{AC}^\star$-Tait is the additional comprehension rule, we get an admissible cut rule for $\mathsf{AC}^\star$-Tait as well. We will only need the following analogue of Corollary 3.4.7.

Proposition 4.4.4. *Assume $\mathsf{AC}^\star$-Tait proves Γ, A and $\Gamma, \neg A$ with w, c-slim proofs of height h and h', respectively. If A is in the Σ-closure of the purely propositional formulae, then $\mathsf{AC}^\star$-Tait proves Γ by a w, c-slim proof of height $h + h'$.*

Theorem 4.4.5. *Let k, n, w, c be natural numbers with $c \cdot w \geq 2$. Assume that there is a w, c-slim $\mathsf{AC}^\star$-proof of Γ of height h where $\Gamma = \Delta, \exists_{4n}\vec{r}\Phi_{n,\ell}(\vec{r})$, with $\Phi_{n,\ell}$ is the n, ℓ-iteration formula. Let η be a partial propositional assignment and $f\colon [2^n] \rightharpoonup [2^n]$ be k-sequential. Assume $|\mathsf{dom}(f)| + cwh < 2^n$. If each element of Δ is either purely propositional or α-free, and $\Delta\eta\eta_f$ is closed, α-free, and false then $\ell \leq k + h$.*

Proof. We argue by induction on h and do case distinction according to the last rule of the proof. Most cases are identical as in the proof of Theorem 4.2.17. So we discuss only the changed ones.

Now, both quantifier rules could be the last rule. The case that the principal formula is $\exists_{4n}\vec{r}\Phi_{n,\ell}(\vec{r})$ remains unchanged. So we may assume that the principal formula is an element of Δ.

Assume the last rule was an $\exists_k$-rule

$$\frac{\Gamma, A(\vec{\wp})}{\Gamma}$$

where, by our assumption, $\exists_k\vec{p}A(\vec{p}) \in \Delta$, hence $\exists_k\vec{p}A(\vec{p})\eta\eta_f$ false. By our assumption on Δ we may conclude that $\exists_k\vec{p}A(\vec{p})$ is, in fact, α-free, since it is not purely propositional. Therefore $\exists_k\vec{p}A(\vec{p})\eta\eta_f = \exists_k\vec{p}A(\vec{p})\eta$. By appropriately extending η, we may assume $A(\vec{\wp})\eta$ to be closed (and, of course, still α-free). But then it has to be false, for otherwise $\exists_k\vec{p}A(\vec{p})\eta$ could not be false. Hence the induction hypothesis applies.

Assume the last rule was an $\forall_k$-rule

$$\frac{\Gamma, A(\vec{a})}{\Gamma}$$

with $\vec{a}$ eigenvariables, without loss of generality not occurring in domain of η. Again, $\forall_k \vec{p} A(\vec{p})\eta\eta_f = \forall_k \vec{p} A(\vec{p})\eta \in \Delta$ and false. But then, for some $\vec{\wp} \in \{\mathrm{T}, \mathrm{F}\}$, $A\eta(\vec{\wp})$ is false. We extend η accordingly and apply the induction hypothesis.

The last remaining case is that of a comprehension rule

$$\frac{\Gamma, \neg(p_1 \leftrightarrow \varphi_1), \ldots, \neg(p_j \leftrightarrow \varphi_j)}{\Gamma}$$

where the φ_i are α-free and the $\vec{p}$ are eigenvariables. Extend η to η'' to make the $\vec{\varphi}\eta''$ closed, without loss of generality not affecting the eigenvariables $\vec{p}$. Now extend η'' to η' by setting $\eta'(p_i)$ to be the truth value of $\varphi_i\eta''$. With this η' we can apply the induction hypothesis. □

Setting $\Delta = \emptyset$, $\eta = \emptyset$, $f = \emptyset$, and $k = 0$, we obtain the usual corollary.

Corollary 4.4.6. *If* $\mathsf{AC}^\star$*-Tait proves* $\exists_{4n}\vec{r}\Phi_{n,\ell}(\vec{r})$ *by a* w, c*-slim proof of height* h *for some* c, w *with* $cw \geq 2$*, then* $h \geq \ell$*.*

5 Theories for Computational Complexity

Bounded Arithmetic has been introduced by Buss [10] in order to make methods of mathematical logic available to computational complexity. He defined first-order theories S^i_2 where the Σ^b_i-definable functions characterise precisely the i'th level of the polynomial hierarchy. Moreover [40], the hierarchy of theories S^i_2 collapses, i.e., $\mathsf{S}_2 = \bigcup_i \mathsf{S}^i_2$ is finitely axiomatisable, if and only if the polynomial-time hierarchy collapses provably in S_2.

The theories S^i_2 are best understood as fragments of Peano Arithmetic (with some appropriate choice of function symbols in the language) with restricted form of induction. Only induction on notation is available in S^i_2 and induction is restricted to formulae is at most i alternations of bounded quantifiers in front of a sharply bounded formula. Induction on notation requires the induction step to go from $A(\lfloor x/2 \rfloor)$ to $A(x)$; in other words, we do induction on the number of bits of x.

As induction on notation is essentially induction up to $\log(x)$, it seems natural to study systems by further restricting the range up to which induction is available to $\log^j(x)$. It should be noted that, with the given choice of language, exponentiation is not available as a total function. Therefore, it does matter how far induction is available.

With the increase of computational power available, the problems that could be tackled in practise grow bigger and bigger. This lead to an increasing interest in smaller complexity classes, like L, NL, AC^k, and NC^k. The latter are often considered as an appropriate model for efficiently parallelisable computation [12, 60, 13]. Most of them have good descriptive characterisations [29, 43, 31, 32, 33] in a setting of finite model theory [42]. In this setting, one considers a finite universe $[n]$, corresponding to the positions in the input and predicates $P_{\mathsf{a}} \subseteq [n]$ corresponding to the letters of the alphabet.

So a natural way to formulate theories corresponding to these complexity classes is to use a language corresponding to this setting. Therefore, a second-order, or, more precisely, two-sorted, language is chosen. Besides a sort for numbers, ranging over positions in the input, a second sort of finite sets, or "bit strings", is present. Quite a few of those two-sorted theories arose [14, 15, 16, 18, 19], mainly to carry out "low-level reverse mathematics" [46], i.e., to study the computational power of certain forms of mathematical reasoning at a very low level.

One of the tools mathematical logic has to offer to analyse formal theories is proof theory [25, 26, 56, 52]. A main theme in proof theory is to determine for which order types a theory proves the principle of transfinite induction [27]. The principle of transfinite induction is naturally formulated

by referring to an uninterpreted relation symbol and therefore quite a few proof-theoretic methods work best for theories having such an uninterpreted predicate available. In a setting where the second sort of "strings" are the main objects of the theory, they cannot simultaneously serve as big unknown predicate. Therefore, one has to define and study relativised theories, building on relativised complexity classes, also studied in Section 2.

In this section we introduce the general setting for Bounded Arithmetic in the two-sorted language and add appropriate relativisation. We also introduce some concrete such theories that we will analyse later.

5.1 The Language of Two-Sorted Bounded Arithmetic

As mentioned in the introduction to this section, we follow the setting of descriptive complexity [33] and our theories will have two sorts, numbers and sets of numbers. We think of the latter as bit strings. This intuition will be very obvious by the way our propositional translation is defined (in Subsection 6.1). The finiteness of the universe in descriptive complexity is reflected in our setting by the length function on sets. The length $|\mathfrak{X}|$ of a set is its length as seen as a string, i.e., one more than the largest element, if $\mathfrak{X}$ is not empty and the length of the empty set is zero.

Some words on the way our relativisation [4] works seem appropriate. First of all, to properly increase the expressive power, the new predicate has to be a property of the "biggest" sort, that is, the string sort. This, however, has some technical implications. Suddenly, strings occur as arguments to something. So it is no longer only important which properties can be expressed, say, by a Δ_0^B-formula (as one would expect for a second order concept), but it also does matter, which string terms we have in our language. Note that in an expression like $\alpha(\mathfrak{X})$ we cannot just simply substitute an arbitrary $\varphi(\cdot)$ for $\mathfrak{X}$.

A consequence of this loss of the substitution property is that we no longer can transform $\forall x < s \exists \mathfrak{X} < t \ldots$ into an $\exists \mathfrak{X}' < t \cdot s \forall x < s \ldots$ as usual. This lack of closure properties and also the need in Subsection 7.1 to properly express the iteration principle (preferably as a strict $\Sigma_1^B(\alpha)$-formula) motivates the addition of one additional string function, the "row function" $\cdot^{[\cdot]}$, which allows one to project to a certain "row" if we think of a string coding a sequence of strings in a matrix-like fashion.

We nevertheless restrict the length function $|\mathfrak{X}|$ to string variables. This is necessary in order to allow a propositional translation (see Subsection 6.1); as the first-order part is coded away—and $|\mathfrak{X}|$ is of the number sort!—every quantification over a string has to include a case distinction about its length. Having only $|\mathfrak{X}|$ in the language, this is not a problem, as there are only

polynomially many possibilities. However, for expressions of the form $|\mathfrak{X}^{[t]}|$ this would no longer be feasible. Consider a string of length at most n^2. It can code n strings of length at most n. This leaves n^n many possible lengths for the various substrings—an expression growing super-polynomially in n.

Even in our restricted setting we still will be able (in $V^0(\alpha)$) to argue as if we had $|\mathfrak{X}^{[t]}|$. Using the Δ_0^B-comprehension axiom, we can obtain a variable $\mathfrak{Y}$ that is equal to $\mathfrak{X}^{[t]}$, so considering $|\mathfrak{Y}|$ will serve the same purpose.

Definition 5.1.1 (The language $\mathcal{L}_2(\alpha)$)**.** The language $\mathcal{L}_2(\alpha)$ is a language in the two sorts

- numbers and
- strings.

Terms, of the number sort, denoted by $r, s, t, \ldots$, are built up form variables $x, y, z, \ldots$, constants 0 and 1, and expressions $|\mathfrak{X}|$ for $\mathfrak{X}$ a string variable, by the binary function symbols $+$ and $\cdot$.

Terms $\mathfrak{T}$ of the string sort are string variables $\mathfrak{X}, \mathfrak{Y}, \mathfrak{Z}, \ldots$, negated string variables $\bar{\mathfrak{X}}, \bar{\mathfrak{Y}}, \bar{\mathfrak{Z}}, \ldots$ or of the form $\mathsf{row}\mathfrak{T}t$ where $\mathfrak{T}$ is a term of the string sort and t a term of the number sort.

Formulae, denoted by $A, B, \ldots$, are built up form the atomic formulae

- $= st$, $\neq st$, $< st$, $\geq st$ for number terms s and t,
- $\mathfrak{T}t$ for $\mathfrak{T}$ a string term and t a number term, and
- $\alpha\mathfrak{T}$ and $\bar{\alpha}\mathfrak{T}$ for $\mathfrak{T}$ a string term

by conjunction $\wedge AB$, disjunction $\vee AB$, bounded number quantification $\forall_< xtA$ and $\exists_< xtA$, bounded string quantification $\forall_< \mathfrak{X}tA$ and $\exists_< \mathfrak{X}tA$, and unbounded quantification $\forall xA$, $\exists xA$, $\forall \mathfrak{X}A$, and $\exists \mathfrak{X}A$.

Quantification $\forall_< xtA$, $\exists_< xtA$, $\forall xA$ and $\exists xA$ binds the (free) occurrences of x in A, but not in t. Quantification $\forall_< \mathfrak{X}tA$, $\exists_< \mathfrak{X}tA$, $\forall \mathfrak{X}A$, and $\exists \mathfrak{X}A$ binds the (free) occurrences of $\mathfrak{X}$ and $\bar{\mathfrak{X}}$ in A. We consider alpha-equivalent formulae as syntactically equivalent.

Remark 5.1.2. Again, syntactical equivalence is denoted by $\equiv$ and the same argument as in Remark 3.1.5 shows that the identification is compatible with our understanding of syntax.

Remark 5.1.3. It should be noted that even though we have more complicated string terms than just the variables, the length function $|\cdot|$ is restricted to variables only. The reason is, that $|\cdot|$ implicitly provides induction and we therefore want to have sets we induct on introduced by an explicit comprehension.

Definition 5.1.4 (Bounded Formulae)**.** An $\mathcal{L}_2(\alpha)$-formula is called *bounded*, if it does not use any unbounded quantification.

Notation 5.1.5. Even though our "official" notation is the Polish one, we use $A \wedge B$, $A \vee B$, $s = t$, $s \neq t$, $s < t$, $s \geq t$, $s \cdot t$, $s + t$, and $\mathfrak{T}^{[t]}$ as abbreviations for $\wedge AB$, $\vee AB$, $= st$, $\neq st$, $< st$, $\geq st$, $\cdot st$, $+st$, and $\mathsf{row}\mathfrak{T}t$, respectively, when there's no danger of confusion. We also use parentheses to disambiguate these abbreviations, or to facilitate reading, as in $\mathfrak{X}(t)$, $\alpha(\mathfrak{X})$. We also write $s > t$ and $s \leq t$ as abbreviations for $t < s$ and $t \geq s$, respectively.

We write $\forall x < t.A$, $\exists x < t.A$, $\forall \mathfrak{X} < t.A$, and $\exists \mathfrak{X} < t.A$ as abbreviations for $\forall_< xtA$, $\exists_< xtA$, $\forall_< \mathfrak{X}tA$, and $\exists_< \mathfrak{X}tA$, respectively.

We write $t \in \mathfrak{X}$ and $\mathfrak{X} \in \alpha$ as a shorthands for $\mathfrak{X}(t)$ and $\alpha(\mathfrak{X})$, respectively.

Remark 5.1.6. Concerning the abbreviations introduced in Notation 5.1.5 and other places, it should be noted that in this thesis we will never code our syntax in any formal theory. Therefore, it does not make much of a difference how we define our language precisely. Nevertheless, in order to get notions like "syntactic equality" well defined, we commit to one definition of our syntax—that in Definition 5.1.1.

Remark 5.1.7. An easy induction shows that all terms of the string sort are of form $\mathfrak{X}^{[t_1]\ldots[t_k]}$ or $\bar{\mathfrak{X}}^{[t_1]\ldots[t_k]}$ for some $k \geq 0$, where $\mathfrak{X}$ a string variable and $t_1, \ldots, t_k$ are terms of the number sort. We often write $\mathfrak{X}^{[\vec{t}]}$ and $\bar{\mathfrak{X}}^{[\vec{t}]}$ for $\mathfrak{X}^{[t_1]\ldots[t_k]}$ and $\bar{\mathfrak{X}}^{[t_1]\ldots[t_k]}$.

Notation 5.1.8. We write $\forall x \leq t.A$ and $\exists x \leq t.A$ as abbreviations for $\forall x < t{+}1.A$ and $\exists x < t{+}1.A$, respectively.

Notation 5.1.9. Addition associates to the left, that is $t_1 + t_2 + \ldots + t_n$ is short for $((t_1 + t_2) + \ldots) + t_n$.

Definition 5.1.10 ($\underline{n}$)**.** If $n \in \mathbb{N}$ is a natural number, by $\underline{n}$ we denote the number term

$$\underbrace{1 + \ldots + 1}_{n}$$

if $n \neq 0$ and set $\underline{0} = 0$.

Remark 5.1.11. Definition 5.1.10 is not the most economic definition of a numeral with respect to term size; using that we have multiplication we could use a binary coding as well.

However, we will never inspect the proof size of arithmetical proofs, only their translations into propositional logic. Hence we don't have to care about the size of numeric terms.

Moreover, the unary coding is more "ideologically correct", as one is supposed to think of numbers (as opposed to strings) as being coded in unary.

Definition 5.1.12 ($\neg A$)**.** The negation $\neg A$ of a formula A is defined by induction on A in the obvious way. In other words

- $\neg(s{=}t) \equiv s{\neq}t$, $\neg(s{\neq}t) \equiv s = t$, $\neg(<st) \equiv \geq st$, $\neg(\geq st) \equiv <st$,
- $\neg\mathfrak{X}^{[\vec{t}]}(t) \equiv \bar{\mathfrak{X}}^{[\vec{t}]}t$, $\neg\bar{\mathfrak{X}}^{[\vec{t}]}(t) \equiv \mathfrak{X}^{[\vec{t}]}(t)$, $\neg(\alpha(\mathfrak{X})) \equiv \bar{\alpha}\mathfrak{X}$, $\neg(\bar{\alpha}\mathfrak{X}) \equiv \alpha(\mathfrak{X})$,
- $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$, $\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$,
- $\neg(\forall_< xtA) \equiv \exists_< xt\neg A$, $\neg(\exists_< xtA) \equiv \forall_< xt\neg A$,
- $\neg(\forall_< \mathfrak{X}tA) \equiv \exists_< \mathfrak{X}t\neg A$, $\neg(\exists_< \mathfrak{X}tA) \equiv \forall_< \mathfrak{X}t\neg A$,
- $\neg(\forall xA) \equiv \exists x\neg A$, $\neg(\exists xA) \equiv \forall x\neg A$,
- $\neg(\forall \mathfrak{X}A) \equiv \exists \mathfrak{X}\neg A$, and $\neg(\exists \mathfrak{X}A) \equiv \forall \mathfrak{X}\neg A$.

Notation 5.1.13. We use $A \to B$ as an abbreviation for $(\neg A)\vee B$. Moreover, $A \leftrightarrow B$ is short for $(A \to B) \wedge (B \to A)$.

Remark 5.1.14. Note that the bounded quantifiers are syntactical entities of their own. However, it will, e.g., be provable that $(\forall x < t.A) \leftrightarrow (\forall x.(x < t \to A))$, as can be seen from Lemma 5.2.2.

Definition 5.1.15 (Ordered Pair $\langle x; y\rangle$)**.** We define the ordered pair of numbers to be

$$\langle x; y\rangle \equiv (x + y)(x + y + 1) + \underline{2} \cdot y \ .$$

Remark 5.1.16. Our definition of the ordered pair deviates slightly from the usual one which would be $2{\cdot}\langle x; y\rangle \equiv (x{+}y)(x{+}y{+}1){+}\underline{2}{\cdot}y$. The reason is, that we want $\langle x; y\rangle$ to be expressible as a *term* in our arithmetical language. For our purpose it doesn't suffice that the *relation* $z = \langle x; y\rangle$ definable by a formula in our language.

Definition 5.1.17 ($\Sigma^B_i(\alpha)$, $\Pi^B_i(\alpha)$)**.** The class $\Sigma^B_0(\alpha) = \Pi^B_0(\alpha)$ is the smallest set of $\mathcal{L}_2(\alpha)$-formulae that contains the atomic formulae and is closed under conjunction, disjunction, and bounded number quantification $\forall_< xtA$, $\exists_< xtA$.

The class $\Sigma^B_{i+1}(\alpha)$ is the smallest set of formulae that contains the $\Pi^B_i(\alpha)$ formulae and is closed under existential bounded string quantification $\exists\mathfrak{X} \leq tA$.

The class $\Pi^B_{i+1}(\alpha)$ is the smallest set of formulae that contains the $\Sigma^B_i(\alpha)$ formulae and is closed under universal bounded string quantification $\forall \mathfrak{X} \leq tA$.

Remark 5.1.18. The reason to use $\exists \mathfrak{X} \leq tA$ in Definition 5.1.17, rather than $\exists \mathfrak{X} < tA$ which might be more suggested by the syntax is, that we want, in the buildup of $\Sigma^B_i(\alpha)$-formula, always to be able to go from a formula A independent of $\mathfrak{X}$ to $\forall \mathfrak{X} \leq tA$ and $\exists \mathfrak{X} \leq tA$. This intuitive property would not hold with strict inequalities.

Remark 5.1.19. Immediately by induction on formulae we note that the negation of a $\Sigma^B_i(\alpha)$-formula is a $\Pi^B_i(\alpha)$-formula, and vice versa.

Definition 5.1.20 ($\Pi^B_\infty(\alpha)$)**.** We set

$$\Pi^B_\infty(\alpha) = \bigcup_{n \in \mathbb{N}} \Pi^B_n(\alpha)$$

to be the set of all formulae that contain bounded quantifiers only.

Remark 5.1.21. Since $\Pi^B_n(\alpha) \subset \Sigma^B_{n+1}(\alpha)$ the set $\Pi^B_\infty(\alpha)$ can also be written as

$$\Pi^B_\infty(\alpha) = \bigcup_{n \in \mathbb{N}} \Sigma^B_n(\alpha)\,.$$

Definition 5.1.22 ($\mathsf{dp}(A)$)**.** If A is an Π^B_∞-formula, then we define its depth $\mathsf{dp}(A)$ by induction on A as follows.

- $\mathsf{dp}(s = t) = \mathsf{dp}(s \neq t) = \mathsf{dp}(s < t) = \mathsf{dp}(s \geq t) = 1$
- $\mathsf{dp}(\mathfrak{T}(t)) = 1$
- $\mathsf{dp}(\alpha(\mathfrak{T})) = 1$
- $\mathsf{dp}(A \wedge B) = \mathsf{dp}(A \vee B) = 1 + \max\{\mathsf{dp}(A), \mathsf{dp}(B)\}$
- $\mathsf{dp}(\forall_< xtA) = \mathsf{dp}(\exists_< xtA) = \mathsf{dp}(\forall_< \mathfrak{X}tA) = \mathsf{dp}(\exists_< \mathfrak{X}tA) = 1 + \mathsf{dp}(A)$

Definition 5.1.23 (Substitution $A[\vec{t}, \vec{\mathfrak{Y}}/\vec{x}, \vec{\mathfrak{X}}]$)**.** If A is a formula, $\vec{x}$ pairwise distinct number variables, $\vec{\mathfrak{X}}$ pairwise distinct string variables, $\vec{t}$ number terms and $\vec{\mathfrak{Y}}$ number variables, we denote by $A[\vec{t}, \vec{\mathfrak{Y}}/\vec{x}, \vec{\mathfrak{X}}]$ the simultaneous, capture-free substitution of $\vec{t}$ for $\vec{x}$ and $\vec{\mathfrak{Y}}$ for $\vec{\mathfrak{X}}$.

Remark 5.1.24. By requiring that the substitution $A[\vec{t}, \vec{\mathfrak{Y}}/\vec{x}, \vec{\mathfrak{X}}]$ be capture free, Definition 5.1.23 is only well-defined up to alpha-equivalence. Fortunately, we identify alpha-equal formulae; see Definition 5.1.1 and Remark 5.1.2.

Remark 5.1.25. In Definition 5.1.23, we only allowed substitution of string variables for string variables and not the more conventional substitution of formulae with distinguished free number variables for string variables. The reason for this is length function $|\cdot|$. Number terms can contain expressions of the form $|\mathfrak{X}|$, but the lengths of arbitrary string terms, let alone formulae, is outside the scope of our language, compare Definition 5.1.1 and Remark 5.1.3. However, for formulae not containing $|\mathfrak{X}|$ or $\alpha(\mathfrak{X}^{[t_1]\dots[t_k]})$ we will define (in Definition 5.1.27) and use the more conventional form of substitution for string variables

Notation 5.1.26. When displaying variables of a formula as in $A(\vec{x}, \vec{\mathfrak{X}})$ this should signify that these variables, among others, may occur in A. This notation does not imply that these variables actually do occur free and the list $\vec{x}, \vec{\mathfrak{X}}$ does not necessarily exhaust all the free variables of A. The purpose of such a display of variables is to distinguish certain variables so that later $A(\vec{t}, \vec{\mathfrak{Y}})$ can be used as a shorthand for the substitution $A[\vec{t}, \vec{\mathfrak{Y}}/\vec{x}, \vec{\mathfrak{X}}]$.

Definition 5.1.27 (Substitution $A[\varphi(x)/\mathfrak{X}]$)**.** Let $A(\mathfrak{X})$ be a formula that neither uses the length of $\mathfrak{X}$, i.e., does not contain $|\mathfrak{X}|$ as a subexpression, nor queries about $\mathfrak{X}$, i.e., does not contain an atom of the form $\alpha(\mathfrak{X}^{[t_1]\dots[t_k]})$. Let $\varphi(x)$ be a formula with a distinguished number variable x. Then $A[\varphi(x)/\mathfrak{X}]$ denotes A with all expressions $\mathfrak{X}^{[t_1]\dots[t_k]}(s)$ replaced by $\varphi(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$ and all expressions $\bar{\mathfrak{X}}^{[t_1]\dots[t_k]}(s)$ replaced by $\neg\varphi(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$.

We adopt Notation 5.1.26 to expressions $A(\varphi)$ accordingly.

Definition 5.1.28 ($\Delta_i^B(\alpha)$-formulae of a theory T)**.** If T is a theory in the language $\mathcal{L}_2(\alpha)$, then a Δ_i^B-formula of T is a pair $\langle\varphi; \psi\rangle$ such that $\varphi \in \Sigma_i^B$ and $\psi \in \Pi_i^B$ and T proves $\varphi \leftrightarrow \psi$.

Remark 5.1.29 ($\Delta_i^B(\alpha)$)**.** Later, we will also refer to just $\Delta_i^B(\alpha)$-formulae without explicitly mentioning a theory. Unless specified otherwise, this will always refer to $\Delta_i^B(\alpha)$-formulae of $V^0(\alpha)$, as defined in Definition 5.2.14. Note that Definition 5.2.14 neither directly nor indirectly refers to the notion of a $\Delta_i^B(\alpha)$-formula.

We write $\Delta_i^B(\alpha)$ to the denote the set of $\Delta_i^B(\alpha)$-formulae (of $V^0(\alpha)$).

Substitution (in the sense of Definition 5.1.23) is extended to $\Delta_i^B(\alpha)$-formulae by always choosing the variant of the two equivalent formulae that

yields the least increase in the logical complexity. This form of substitution is the main benefit of considering $\Delta_i^B(\alpha)$-formulae and will be made precise in Lemma 5.1.32.

Definition 5.1.30 (Substitution $A[\varphi(x)/\mathfrak{X}]$ for $\varphi(x)$ a $\Delta_i^B(\alpha)$-formula)**.** Let $A(\mathfrak{X})$ be a formula that neither uses the length of $\mathfrak{X}$, i.e., does not contain $|\mathfrak{X}|$ as a subexpression, nor queries about $\mathfrak{X}$, i.e., does not contain an atom of the form $\alpha(\mathfrak{X}^{[t_1]\dots[t_k]})$. Let $\varphi(x) = \langle\psi(x); \psi'(x)\rangle$ be a $\Delta_i^B(\alpha)$-formula of some theory T in the language $\mathcal{L}_2(\alpha)$ with a distinguished number variable x. Then $A[\varphi(x)/\mathfrak{X}]$ denotes A with all expressions $\mathfrak{X}^{[t_1]\dots[t_k]}(s)$ replaced by $\psi(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$ or $\psi'(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$, and all expressions $\bar{\mathfrak{X}}^{[t_1]\dots[t_k]}(s)$ replaced by $\neg\psi'(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$ or $\neg\psi(\langle t_1; \langle \dots; \langle t_k; s\rangle\rangle\rangle)$.

Here, the first option is chosen in positions where the innermost string-quantifier they are in the scope of is existential, or in positions that are not under the scope of any string quantification. Otherwise, the second choice is taken.

We adopt Notation 5.1.26 accordingly to expressions $A(\varphi)$, for φ a $\Delta_i^B(\alpha)$-formula of some theory (understood from the context).

Notation 5.1.31 (Δ_i^B)**.** We use expression Δ_i^B-formula (of some theory T), to denote $\Delta_i^B(\alpha)$-formulae not containing α.

The main point about Δ_1^B-formula is that, in a context where at least one quantifier is present, they can be used as if they where Π_0^B. This is made formal in the next lemma.

Lemma 5.1.32. *If $A(\mathfrak{X})$ is a $\Sigma_{i+1}^B(\alpha)$-formula for some $i \geq 0$ and $\varphi(x)$ is a $\Delta_1^B(\alpha)$ such that $A(\varphi)$ is defined, then $A(\varphi)$ is still $\Sigma_{i+1}^B(\alpha)$.*

Proof. First, a simple induction on B shows that, if $B(\mathfrak{X})$ is $\Pi_0^B(\alpha)$ then $B(\varphi)$ can be both, $\Sigma_1^B(\alpha)$ and $\Pi_1^B(\alpha)$. Then, induction on i yields the claim. □

5.2 The Basic Theory $V^0(\alpha)$

Definition 5.2.1 (Two-Sorted predicate Logic)**.** *Two-sorted predicate logic* is classical first-order logic, restricted to the language $\mathcal{L}_2(\alpha)$, this is, requiring that the sorts be respected. We assume a Tait-style formalisation where in the description of the rule we omit side formulae; we add the rules for the bounded quantifiers as following.

$$\frac{a \geq t, A[a/x]}{\forall x < t.A} \qquad \frac{s < t \wedge A[s/x]}{\exists x < t.A}$$

$$\frac{|\mathfrak{Y}| \geq t, A[\mathfrak{Y}/\mathfrak{X}]}{\forall \mathfrak{X} < t.A} \qquad \frac{|\mathfrak{Y}| < t \wedge A[\mathfrak{Y}/\mathfrak{X}]}{\exists \mathfrak{X} < t.A}$$

Here in the universal rules we assume the eigenvariable condition for a and $\mathfrak{Y}$; for the existential rules s and $\mathfrak{Y}$ may be arbitrary.

Lemma 5.2.2. *The following are consequences of two-sorted predicate logic and can be proved in a cut-free way.*

- $(\forall x < t.A) \leftrightarrow (\forall x(x < t \to A))$
- $(\exists x < t.A) \leftrightarrow (\exists x(x < t \wedge A))$
- $(\forall \mathfrak{X} < t.A) \leftrightarrow (\forall X(|\mathfrak{X}| < t \to A))$
- $(\exists \mathfrak{X} < t.A) \leftrightarrow (\exists X(|\mathfrak{X}| < t \wedge A))$

Proof. For example the first equivalence is proved as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a < t \wedge \neg A[a/x],\ a \geq t,\ A[a/x]}}{\exists x < t.\neg A,\ a \geq t,\ A[a/x]}}{\exists x < t.\neg A,\ a \geq t \vee A[a/x]}}{\exists x < t.\neg A,\ \forall x(x \geq t \vee A)} \quad \text{and} \quad \dfrac{\dfrac{\dfrac{}{a < t \wedge \neg A[a/x],\ a \geq t,\ A[a/x]}}{\exists x(x < t \wedge \neg A),\ a \geq t,\ A[a/x]}}{\exists x(x < t \wedge \neg A),\ \forall x < t.A}$$

The proofs for the remaining three equivalences are very similar. □

Definition 5.2.3 (Basic Axioms)**.** The *basic axioms of two-sorted arithmetical theories* are all substitution instances of the following formulae

- $x + 1 \neq 0$ and $x + 1 = y + 1 \to x = y$ and $x = 0 \vee \exists y < x.x = y + 1$
- $x + 0 = x$ and $x + (y + 1) = (x + y) + 1$ and $0 + 1 = 1$
- $x \cdot 0 = 0$ and $x \cdot (y + 1) = x \cdot y + x$
- $x \geq 0$ and $x + y \geq x$
- $x < y \vee y < x \vee x = y$
- $y < x \vee z < y \vee z \geq x$
- $x \geq y \vee y \geq x$
- $y < x \vee x < y + 1$ and $x \geq y + 1 \vee y \geq x$
- $\mathfrak{X}(y) \to y < |\mathfrak{X}|$ and $y + 1 = |\mathfrak{X}| \to \mathfrak{X}(y)$

and the equality axioms as follows.

- $r = r$ and $s = t \to t = s$ and $r = s \to s = t \to r = t$

- $t = t' \to r[t/x] = r[t'/x]$
- $t = t' \to \mathfrak{X}(t) \to \mathfrak{X}(t')$

Remark 5.2.4. The basic axioms are the basic axioms B1–B14 and L1–L2 of Cook's formulation [14] of V^0. However, some abbreviations have been expanded to show the actual symbols $<$ and $\geq$ of our official language.

Proposition 5.2.5. *The basic axioms imply* $x < y + 1 \leftrightarrow x \leq y$.

Proof. Both conjuncts of the conjunctive normal form of this statement are basic axioms. □

Proposition 5.2.6. *The basic axioms imply* $\mathfrak{X}(x) \to |\mathfrak{X}| > x$.

Proof. This is one of the basic axioms. □

Notation 5.2.7. We write $\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}$ as abbreviation for

$$(\forall i < |\mathfrak{X}|.\mathfrak{X}^{[\vec{s}]}(i) \leftrightarrow \mathfrak{Y}^{[\vec{t}]}(i)) \wedge (\forall i < |\mathfrak{Y}|.\mathfrak{X}^{[\vec{s}]}(i) \leftrightarrow \mathfrak{Y}^{[\vec{t}]}(i))$$

and use similar abbreviations for strings ending in a negated variable $\bar{\mathfrak{X}}$.

Proposition 5.2.8. *Equality of strings is symmetric, that is,* $\mathfrak{T} = \mathfrak{T}' \to \mathfrak{T} = \mathfrak{T}'$ *is a consequence of the equality axioms.*

Proof. Expanding the abbreviation $\mathfrak{T} = \mathfrak{T}'$ and using the characterisation of Lemma 5.2.2 of the bounded quantifiers, the claim follows purely logically. □

Proposition 5.2.9. *If* $\mathfrak{X} = \mathfrak{Y}$ *then* $|\mathfrak{X}| = |\mathfrak{Y}|$.

Proof. Assume $\mathfrak{X} = \mathfrak{Y}$ and, for the sake of contradiction, $|\mathfrak{X}| \neq |\mathfrak{Y}|$. Without loss of generality $|\mathfrak{X}| < |\mathfrak{Y}|$. In particular $|\mathfrak{Y}| \neq 0$, so there is a y such that $y + 1 = |\mathfrak{Y}|$. But then $\mathfrak{Y}(y)$ and therefore, since $y \leq |\mathfrak{Y}|$, we have $\mathfrak{X}(y)$ which, in turn, implies $y < |\mathfrak{X}|$. But then $|\mathfrak{Y}| = y + 1 \leq |\mathfrak{X}|$, contradicting $|\mathfrak{Y}| > |\mathfrak{X}|$. □

Equality of strings can be shown extensionally.

Proposition 5.2.10.

$$(\forall i.(\mathfrak{T}(i) \leftrightarrow \mathfrak{T}'(i))) \to \mathfrak{T} = \mathfrak{T}$$

is a purely logical consequence.

Proof. We use the characterisation of Lemma 5.2.2 of bounded quantifiers and note that $\forall i\varphi$ implies $\forall i < t.\varphi$. Hence the claim is immediate by unfolding the abbreviations. □

The converse is immediate for string variables.

Proposition 5.2.11. *From the basic axioms the following equality axiom*

$$\mathfrak{X} = \mathfrak{Y} \to \mathfrak{X}(i) \to \mathfrak{Y}(i)$$

is derivable.

Proof. Argue informally in the theory of the basic axioms.

Assume $\mathfrak{X} = \mathfrak{Y}$ and $\mathfrak{X}(i)$. From $\mathfrak{X}(i)$ we get $i < |\mathfrak{X}|$, hence we can eliminate the bounded quantifier and get $\mathfrak{Y}(i)$. □

There is one equality axiom for strings missing, which we have to assert in the respective theories.

Definition 5.2.12 (String Extensionality)**.** By "string extensionality" we mean the following axiom.

$$\mathfrak{T} = \mathfrak{T}' \to \alpha(\mathfrak{T}) \to \alpha(\mathfrak{T}')$$

We have not formulated an equality axiom for the row function. The reason is, that the row function has an explicit definition and so the equality axiom is derivable anyway.

Note, however, that the explicit definition does not render the row function useless. The presence of the row function provides our theories with more terms of the string sort, and therefore, in particular, with more arguments we can give to the parameter α.

Definition 5.2.13 (Row Axiom)**.** The *row axiom* is the axiom

$$\mathfrak{T}^{[t]}(s) \leftrightarrow \mathfrak{T}(\langle t; s\rangle)$$

Definition 5.2.14 ($V^0(\alpha)$)**.** The system $V^0(\alpha)$ is defined to be the basic axioms, the row axiom and string extensionality with two-sorted predicate logic in the language $\mathcal{L}_2(\alpha)$, and the $\Sigma^B_0(\alpha)$-comprehension axioms

$$\exists \mathfrak{Z} < y + 1 \forall i < y(\mathfrak{Z}(i) \leftrightarrow \varphi(i, \vec{x}, \vec{\mathfrak{X}}))$$

where $\varphi(i, \vec{x}, \vec{\mathfrak{X}})$ is any $\Sigma^B_0(\alpha)$-formula and $\mathfrak{Z}$ a fresh variable (i.e., not contained in φ).

The next few lemmata are a step towards the proof of the induction principle in Lemma 5.2.18. The presentation follows closely Cook's manuscript [14].

Lemma 5.2.15. *$V^0(\alpha)$ proves the following minimisation principle.*

$$\mathfrak{X}(y) \to \exists x \leq y(\mathfrak{X}(x) \wedge \forall z < x \neg \mathfrak{X}(z))$$

Proof. Argue informally in $V^0(\alpha)$.

We use Σ^B_0-comprehension to obtain a $\mathfrak{Z}$ with $|\mathfrak{Z}| < y + 1$ and

$$\forall i < y(\mathfrak{Z}(i) \leftrightarrow \forall z \leq i \neg \mathfrak{X}(z))$$

Let $x = |\mathfrak{Z}|$. We want to show $\mathfrak{X}(x)$ and $\forall z < x \neg \mathfrak{X}(z)$ which shows that x is the desired witness.

First consider the case $x = 0$. Then the second claim holds true vacuously. To establish $\mathfrak{X}(0)$ note that $\neg\mathfrak{X}(0)$ would imply $\mathfrak{Z}(0)$ and therefore $|\mathfrak{Z}| > 0$.

If $x \neq 0$ then $x = x' + 1$ for some x'. Hence we obtain $\mathfrak{Z}(x')$ and $\neg\mathfrak{Z}(x)$ from the basic axioms on the length of a string. This leaves us with the situation

$$\forall z \leq x' \neg \mathfrak{X}(z) \qquad \text{and} \qquad \exists z \leq x' + 1 \mathfrak{X}(z)$$

Using that the basic axioms enforce $z \leq x' + 1 \to z \leq x' \vee z = x' + 1$ the claim follows by logical reasoning. □

Remark 5.2.16. In the proof of Lemma 5.2.15 it was essential that Σ^B_0 is closed under bounded number quantification.

Lemma 5.2.17. *$V^0(\alpha)$ proves the following set-induction axiom.*

$$[\mathfrak{X}(0) \wedge \forall x < z(\mathfrak{X}(x) \to \mathfrak{X}(x + 1))] \to \mathfrak{X}(z)$$

Proof. Argue informally in $V^0(\alpha)$.

Assume $\mathfrak{X}(0) \wedge \forall x < z(\mathfrak{X}(x) \to \mathfrak{X}(x+1))$. We have to show $\mathfrak{X}(z)$. By Σ^B_0-comprehension we obtain a $\mathfrak{Y}$ with $|\mathfrak{Y}| \leq z + 1$ and $\forall i \leq z(\mathfrak{Y}(i) \leftrightarrow \neg\mathfrak{X}(i))$.

Suppose $\neg\mathfrak{X}(z)$ to show a contradiction. This implies $\mathfrak{Y}(z)$ and by set-minimisation (Lemma 5.2.15) we obtain an $x \leq z$ with $\mathfrak{Y}(x) \wedge \forall i < x \neg\mathfrak{Y}(i)$. If $x = 0$ then $\mathfrak{Y}(0)$ contradicts the assumption $\mathfrak{X}(0)$. If $x \neq 0$ then $x = y + 1$ for some y and we $\neg\mathfrak{Y}(y) \wedge \mathfrak{Y}(y + 1)$ which contradicts $\forall x < z(\mathfrak{X}(x) \to \mathfrak{X}(x + 1))$. □

Lemma 5.2.18. *$V^0(\alpha)$ proves the following bounded $\Sigma^B_0(\alpha)$-induction axiom.*

$$[\varphi(0) \wedge \forall x < z(\varphi(x) \to \varphi(x + 1))] \to \varphi(z)$$

where φ is a $\Sigma^B_0(\alpha)$-formula, possibly with parameters of both sorts.

Proof. Argue informally $V^0(\alpha)$.

By $\Sigma^B_i(\alpha)$-comprehension we obtain an $\mathfrak{X}$ with $|\mathfrak{X}| \leq z+1$ and $\forall i \leq z(\mathfrak{X}(i) \leftrightarrow \varphi(i))$. Assume $\varphi(0) \wedge \forall x < z(\varphi(x) \rightarrow \varphi(x+1))]$. Then, by the properties of $\mathfrak{X}$ we obtain $\mathfrak{X}(0) \wedge \forall x < z(\mathfrak{X}(x) \rightarrow \mathfrak{X}(x+1))]$, and $\mathfrak{X}$-induction (Lemma 5.2.17) gives us $\mathfrak{X}(z)$, hence $\varphi(z)$. □

Corollary 5.2.19. *$V^0(\alpha)$ proves the following $\Sigma^B_0(\alpha)$-Induction axiom.*

$$[\varphi(0) \wedge \forall x(\varphi(x) \rightarrow \varphi(x+1))] \rightarrow \forall z \varphi(z)$$

where φ is a $\Sigma^B_0(\alpha)$-formula, possibly with parameters of both sorts.

Proof. Follows from Lemma 5.2.18 by first order logic. □

Having now enough induction available, we prove some basic arithmetical facts. They will, in particular, provided the needed monotonicity properties of our coding, in particular of $\langle x; y\rangle$.

Proposition 5.2.20. *As a theorem of $V^0(\alpha)$, it is the case that $x+y \geq x$ and $x+y \geq y$.*

Proof. Argue informally in $V^0(\alpha)$.

$x+y \geq x$ is one of the basic axioms. We show $x+y \geq y$ by induction on y. If $y=0$, then $x+0 \geq 0$ is one of the basic axioms. If $y = y'+1$ then $x+y = x+(y'+1) = (x+y')+1$. By induction hypothesis $x+y' \geq y'$, and so by Proposition 5.2.5, we have $(x+y')+1 > y'$. We have to show $(x+y')+1 \geq y'+1$; assuming the contrary we get $(x+y')+1 < y'+1$, which by Proposition 5.2.5 again, is the same as saying $(x+y')+1 \leq y'$. But his contradicts $(x+y')+1 > y'$. □

Proposition 5.2.21. *$V^0(\alpha)$ proves that $\langle x; y\rangle \geq y$.*

Proof. Arguing informally in $V^0(\alpha)$ we note that $\langle x; y\rangle = \cdots + \underline{2} \cdot y \geq \underline{2} \cdot y = y+y \geq y$, where for the first inequality we used Proposition 5.2.20. □

We now show that all the remaining extensionality properties hold as a theorem of $V^0(\alpha)$.

Proposition 5.2.22. *As a theorem of $V^0(\alpha)$ the equality axiom*

$$\mathfrak{T} = \mathfrak{T}' \rightarrow \mathfrak{T}(i) \rightarrow \mathfrak{T}'(i)$$

holds.

Proof. Argue informally in $V^0(\alpha)$.

Assume $\mathfrak{T} = \mathfrak{T}'$ and $\mathfrak{T}(i)$. By Remark 5.1.7 we know that $\mathfrak{T}$ is of the form $\mathfrak{T} \equiv \mathfrak{X}^{[\vec{t}]}$. By the row axiom from $\mathfrak{T}(i)$ we therefore get $\mathfrak{X}(\langle \ldots; \langle \ldots; i\rangle\rangle)$ and therefore $\langle \ldots; \langle \ldots; i\rangle\rangle < |\mathfrak{X}|$. Iterated application of Proposition 5.2.21 yields $i \leq \langle \ldots; \langle \ldots; i\rangle\rangle < |\mathfrak{X}|$. So we can eliminate the bounded quantifier and get $\mathfrak{T}'(i)$. □

Proposition 5.2.23. *The following is a theorem of* $V^0(\alpha)$.

$$\mathfrak{T} = \mathfrak{T}' \to t = t' \to \mathfrak{T}^{[t]} = \mathfrak{T}'^{[t']}$$

Proof. Argue informally in $V^0(\alpha)$.

Assume $\mathfrak{T} = \mathfrak{T}'$ and $t = t'$. We want to show $\mathfrak{T}^{[t]} = \mathfrak{T}^{[t']}$. By Proposition 5.2.10 it suffices to show $\forall i(\mathfrak{T}^{[t]}(i) \leftrightarrow \mathfrak{T}'^{[t']}(i))$. By the row axiom, it suffices to show $\forall i(\mathfrak{T}(\langle t; i\rangle) \leftrightarrow \mathfrak{T}'(\langle t'; i\rangle)$. So let i be given. From $t = t'$ and the equality axioms we obtain $\langle t; i\rangle = \langle t'; i\rangle$. Since we have $\mathfrak{T} = \mathfrak{T}'$, the claim $\mathfrak{T}(\langle t; i\rangle) \leftrightarrow \mathfrak{T}(\langle t'; i\rangle)$ follows from Proposition 5.2.22. □

Proposition 5.2.24. *The basic axioms together with the row axioms imply*

$$\mathfrak{X} = \mathfrak{Y} \to t(\mathfrak{X}) = t(\mathfrak{Y})$$

for any $\mathcal{L}_2(\alpha)$*-term* t.

Proof. Induction on t. The only non-trivial case is when $t(\mathfrak{X}) \equiv |\mathfrak{X}|$ and $t(\mathfrak{Y}) \equiv |\mathfrak{Y}|$. Here Proposition 5.2.9 yields the claim. □

Lemma 5.2.25. *If* $\varphi(\mathfrak{X})$ *is a* $\mathcal{L}_2(\alpha)$*-formula, then*

$$\mathfrak{X} = \mathfrak{Y} \to (\varphi(\mathfrak{X}) \leftrightarrow \varphi(\mathfrak{Y}))$$

is a theorem of $V^0(\alpha)$.

Proof. Induction on φ. For the "numeric" atomic formulae, that is, if φ is of one of the forms $s = t$, $s \neq t$, $s < t$ or $s \geq t$, then Proposition 5.2.24 provides the needed equalities.

For $\varphi(\mathfrak{X}) \equiv \mathfrak{X}^{[\vec{s}]}(t)$ we first have $t[\mathfrak{Y}/\mathfrak{X}] = t$ and $\vec{s}[\mathfrak{Y}/\mathfrak{X}] = \vec{s}$ again by Proposition 5.2.24. Iterated application of Proposition 5.2.23 yields $\mathfrak{X}^{[\vec{s}]} = (\mathfrak{X}^{[\vec{s}]})[\mathfrak{Y}/\mathfrak{X}]$, hence Proposition 5.2.22 yields the claim $\mathfrak{X}^{[\vec{s}]}(t) \leftrightarrow (\mathfrak{X}^{[\vec{s}]}(t))[\mathfrak{Y}/\mathfrak{X}]$.

The case $\varphi(\mathfrak{X}) \equiv \bar{\mathfrak{X}}^{[\vec{s}]}(t)$ is handled similarly. To show $\mathfrak{X} = \mathfrak{Y} \to \alpha(\mathfrak{T}) \leftrightarrow \alpha(\mathfrak{T}[\mathfrak{Y}/\mathfrak{X}])$ we again first obtain $\mathfrak{T} = \mathfrak{T}[\mathfrak{Y}/\mathfrak{X}]$ from $\mathfrak{X} = \mathfrak{Y}$; then we can use the string extensionality axiom.

If φ is not an atomic formula, the claim follows immediately from the induction hypotheses by the fact that logical equivalence is a congruence relation with respect to logical connectives. □

5.3 The Theories $\mathsf{VL}(\alpha)$ and $\mathsf{VNL}(\alpha)$

Motivated by Proposition 2.5.14, we know define a theory $\mathsf{VNL}(\alpha)$ for NL^α by axiomatising that the problem of the reachability relation has a solution. Using the pairing of natural numbers, a set variable $\mathfrak{X}$ can be considered a directed graph on $[a]$ by defining the directed edge from i to j to be present if and only if $\mathfrak{X}\langle i; j\rangle$.

With this reading in mind, it is easy to define a Δ_0^B-formula asserting that $\mathfrak{Y}\langle t; i\rangle$ holds if and only if there is a path of length at most t from 0 to i.

Definition 5.3.1 ($\delta_{\text{conn}}(a, \mathfrak{X}, \mathfrak{Y})$)**.** The connectivity formula $\delta_{\text{conn}}(a, \mathfrak{X}, \mathfrak{Y})$ is defined to be

$$\begin{aligned}&\mathfrak{Y}\langle 0; 0\rangle \wedge \forall x < a(x \neq 0 \rightarrow \neg\mathfrak{Y}\langle 0; x\rangle)\wedge\\ &\forall z < a \forall x < a\,[\mathfrak{Y}\langle z{+}1; x\rangle \leftrightarrow (\mathfrak{Y}\langle z; x\rangle \vee \exists y < a(\mathfrak{Y}\langle z; y\rangle \wedge \mathfrak{X}\langle y; x\rangle))]\ .\end{aligned}$$

Definition 5.3.2 ($\mathsf{VNL}(\alpha)$)**.** The theory $\mathsf{VNL}(\alpha)$ is defined to be $V^0(\alpha)$ plus the axiom

$$\exists \mathfrak{Y} < \langle a; a\rangle \delta_{\text{conn}}(a, \mathfrak{X}, \mathfrak{Y})$$

For L^α the corresponding complete problem is s-t-connectivity for graphs with out-degree at most one. This can be expressed by saying that, if j and j' can both be reached by an edge from i, then $j = j'$.

Definition 5.3.3 ($\mathsf{VL}(\alpha)$)**.** The theory $\mathsf{VL}(\alpha)$ is defined to be $V^0(\alpha)$ plus the axiom

$$[\forall i, j, j' < a(\mathfrak{X}\langle i; j\rangle \rightarrow \mathfrak{X}\langle i; j'\rangle \rightarrow j = j')]\ \rightarrow\ \exists \mathfrak{Y} < \langle a; a\rangle\, \delta_{\text{conn}}(a, \mathfrak{X}, \mathfrak{Y})$$

Immediately from the definition we note that $\mathsf{VL}(\alpha) \subseteq \mathsf{VNL}(\alpha)$.

5.4 The Logarithm Function

As discussed in the introduction, one natural way to obtain further theories of Bounded Arithmetic that are of interest in the literature is to restrict the range of induction to $\log^j(x)$. It order to formally define this, we need a notion of logarithm in the language. An obvious, and legitimate, way to achieve this, is to add a new function symbol. However, this is not necessary, as the relation "$n = 2^m$" is already AC^0-definable.

Mainly following a standard text book [33], we show that the bit relation can be defined, that is, the relation saying that the i'th bit of the representation of j in binary is one. Another presentation of an exponentiation function can be found in the new text book by Cook and Nguyen [18].

As an application, we can convert back and forth between numbers and their string representation. We will also define a logarithm relation saying $i = |j|$; this will allow us to define in Subsection 5.5 systems $V^{i/j}(\alpha)$ with restricted induction on more complex formulae than just the $\Sigma^B_0(\alpha)$.

This subsection is quite technical and essentially known in the literature. It can be safely skipped assuming that we had a relation symbol $\mathrm{EXP}(i, j)$ expressing $2^i = j$ with the defining equations added as axioms.

Moreover, adding additional true α-free first-oder axioms will not change any of the strength measures considered. In fact, Proposition 6.4.1 will show that heights of proofs translated into propositional logic—and therefore also the strength measure we will build on this—are insensitive to the addition of true purely first-order axioms. Therefore, we will be a bit sketchy in this section about properties provable in $V^0(\alpha)$. All the properties needed are of the right shape, i.e., true and purely first-order. Therefore, they could as well be added as additional axioms instead of showing that they are already theorems of $V^0(\alpha)$. This would not change the sequel of this work in any way.

Definition 5.4.1 ($\mathrm{DIVIDES}(x, y)$)**.** We define the abbreviation

$$\mathrm{DIVIDES}(x, y) \equiv \exists z \leq y(z \cdot x = y)$$

saying that x divides y.

Proposition 5.4.2. $\mathrm{DIVIDES}(x, y) \in \Pi^B_0$

Proof. By inspection of the definition. □

Proposition 5.4.3. $\mathbb{N} \models \mathrm{DIVIDES}(\underline{n}, \underline{m})$ *if and only if* n *is a divisor of* m*.*

Proof. By the definition of the divisor relation in $\mathbb{N}$. □

Proposition 5.4.4. *For* $n \in \mathbb{N}$ *the following is a consequence of the basic axioms.*

$$x \cdot \underline{n} = \underbrace{x + \ldots + x}_{n}$$

Proof. (Meta)induction on n. □

Remark 5.4.5. Note that in Proposition 5.4.4 it was essential that the numeral was multiplied "from the right".

Lemma 5.4.6. *If* $n \in \mathbb{N}$ *then the following is a consequence of the basic axioms.*

$$\mathrm{DIVIDES}(\underline{n}, x) \leftrightarrow \exists z \leq x(\underbrace{z + \ldots + z}_{n} = x)$$

Proof. Immediately from the Definition 5.4.1 and Proposition 5.4.4. □

Definition 5.4.7 ($\mathrm{POW}_2(x)$)**.** We define the abbreviation

$$\mathrm{POW}_2(x) \equiv \forall y < x.\mathrm{DIVIDES}(y, x) \wedge y \geq \underline{2} \rightarrow \mathrm{DIVIDES}(\underline{2}, y)$$

saying that x is a power of 2.

Proposition 5.4.8. $\mathrm{POW}_2(x) \in \Pi_0^B$

Proof. By inspection of Definition 5.4.7 and Propositions 5.4.2. □

Proposition 5.4.9. $\mathbb{N} \models \mathrm{POW}_2(\underline{n})$ *if and only if* n *is a power of two.*

Proof. Immediately from Definition 5.4.7 and Propositions 5.4.3 we see that every power of two n has the property $\mathrm{POW}_2(\underline{n})$.

If, on the other hand, n is not a power of two, than there is some prime $p > 2$, that divides n. However 2 is then not a divisor of p. □

Notation 5.4.10. If $n = \sum_i 2^i \cdot a_i$ with $a_i \in \{0, 1\}$ then it is well known that the a_i are uniquely determined and we denote a_i by $n[i]$ and call it the "i'th bit of the binary representation of n".

For example $13[0] = 1$, $13[1] = 0$, $13[2] = 1$, and $13[3] = 1$.

Definition 5.4.11 ($\mathrm{BIT}'(x, y)$)**.** We define the abbreviation

$$\mathrm{BIT}'(x, y) \equiv \mathrm{POW}_2(y) \wedge (\exists u < x)(\exists v < y)(x = \underline{2} \cdot u \cdot y + y + v)$$

expressing that, for some i, we have $y = 2^i$ and the i'th bit of the binary representation of x is 1.

Proposition 5.4.12. $\mathrm{BIT}'(x, y) \in \Pi_0^B$

Proof. By Inspection of Definition 5.4.11 and Propositions 5.4.8. □

Proposition 5.4.13. $\mathbb{N} \models \mathrm{BIT}'(\underline{n}, \underline{k})$ *if and only if* k *is of the form* $k = 2^i$ *and* $n[i] = 1$.

Proof. By Proposition 5.4.9 we know that k is a power of two, that is $k = 2^i$, for some $i \in \mathbb{N}$. Moreover, $n = 2uk + k + v$ for some u and $v < k$. Therefore $n = u \cdot 2^{i+1} + 2^i + v$ with $v < 2^i$, hence the i'th bit is set.

On the other hand, if $k = 2^i$ and the i'th bit n is set, we can chose $u = \lfloor n/2^{i+1} \rfloor$ and $v = n \bmod 2^{i+1}$. □

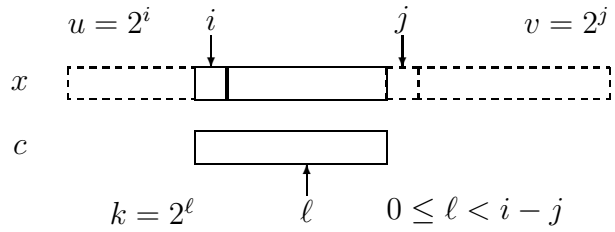

Figure 1: The specification of BITCOPY(x, u, v, c).

Definition 5.4.14 (BITCOPY(x, u, v, c))**.** We define the abbreviation

$$\begin{aligned}&\mathrm{BITCOPY}(x, u, v, c) \equiv \\ &(\forall k \leq c)(\mathrm{POW}_2(k) \rightarrow v \cdot k < u \ \wedge \ (\mathrm{BIT}'(c, k) \leftrightarrow \mathrm{BIT}'(x, \underline{2} \cdot v \cdot k)))\end{aligned}$$

expressing that for $u = 2^i$ and $v = 2^j$, then the binary representation of c consists of bits $i \ldots (j + 1)$ of x.

The specification of BITCOPY(x, u, v, c) is illustrated in Figure 1.

Proposition 5.4.15. $\mathrm{BITCOPY}(x, u, v, c) \in \Pi^B_0$

Proof. Immediately by inspection of Definition 5.4.14, using Propositions 5.4.8 and 5.4.12. □

Proposition 5.4.16. *Suppose that $i \geq j$ are natural numbers. Then* $\mathbb{N} \models \mathrm{BITCOPY}(\underline{n}, \underline{2^i}, \underline{2^j}, \underline{m})$ *if and only if* $(\forall 0 \leq \ell < i - j)(m[\ell] = n[\ell + j + 1])$ *and* $(\forall \ell \geq i - j)(m[\ell] = 0)$.

Proof. Using Propositions 5.4.9 and 5.4.11 for the soundness of $\mathrm{POW}_2(\cdot)$ and $\mathrm{BIT}'(\cdot, \cdot)$ we can rewrite $\mathbb{N} \models \mathrm{BITCOPY}(\underline{n}, \underline{2^i}, \underline{2^j}, \underline{m})$ equivalently as

$$\forall \ell (2^\ell \leq m \rightarrow 2^j \cdot 2^\ell < 2^i \wedge (m[\ell] \leftrightarrow n[1 + j + \ell]))$$

Using that $2^j \cdot 2^\ell < 2^i$ is equivalent to $\ell < i - j$, and $2^\ell \leq m$ is equivalent to $(\exists \ell' \geq \ell)(m[\ell'] = 1)$. This yields the claim. □

Remark 5.4.17. The intuitive meaning of EXPCOMP(a, b) is that, if i and j are consecutive set bits of a, then bits $i \ldots (j + 1)$ of b are the binary representation of i. In other words, EXPCOMP(a, b) says that a and b code a computation of 2^i by "repeated squaring", that is, by using

$$2^{2 \cdot i} = (2^i)^2 \qquad \text{and} \qquad 2^{2 \cdot i + 1} = 2 \cdot (2^i)^2 \, .$$

Since we must fit in the binary representation of i into $i - j$ many bits, this leaves us with the base cases 3, 4, and 5 for i.

Figure 2 shows an example of numbers a and b, represented in binary, for which EXPCOMP(a, b) holds. Figure 3 shows the base cases.

	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a	1	0	0	0	0	0	0	1	0	0	1	0	1	0
b	0	0	0	1	1	0	1	1	1	0	1	1	0	0

Figure 2: The computation of 2^{13}. Note that the binary representation of 13 is 1101.

3	2	1	0
1	0	1	
1	1		

4	3	2	1	0
1	0	0	1	
1	0	0		

5	4	3	2	1	0
1	0	0	0	1	
0	1	0	1		

Figure 3: The base cases for EXPCOMP$(\cdot, \cdot)$.

Definition 5.4.18 (CONSEC(a, u, v))**.** We define the abbreviation

$$\begin{aligned}&\mathrm{CONSEC}(a, u, v) \equiv \\ &u > v \wedge \mathrm{BIT}'(a, u) \wedge \mathrm{BIT}'(a, v) \wedge (\forall k < u)(k > v \rightarrow \neg \mathrm{BIT}'(a, k))\end{aligned}$$

expressing that u and v name consecutive bits in a.

Proposition 5.4.19. $\mathrm{CONSEC}(x, u, v)c \in \Pi_0^B$

Proof. By inspection of Definition 5.4.18, using Proposition 5.4.12. □

Proposition 5.4.20. $\mathbb{N} \models \mathrm{CONSEC}(\underline{n}, \underline{2^i}, \underline{2^j})$ *if and only if* i *and* j *are consecutive bits of* n*, that is, if* $i > j$ *and* $n[i] = 1$ *and* $n[j] = 1$ *and* $(\forall i < \ell < j)(n[\ell] = 0)$*.*

Proof. This follows immediately from Proposition 5.4.13 □

Definition 5.4.21 (EXPCOMP(a, b))**.** We define the abbreviation

$$\begin{aligned}&\mathrm{EXPCOMP}(a, b) \equiv \\ &\mathrm{EXPCOMPinit}(a, b) \wedge \\ &(\forall u \leq a)(\forall k \leq a)(\forall v \leq a)((\mathrm{CONSEC}(a, u, k) \wedge \mathrm{CONSEC}(a, k, v)) \rightarrow \\ &\quad ((\exists i \leq b)(\exists j < b)(\mathrm{BITCOPY}(b, u, k, i) \wedge \mathrm{BITCOPY}(b, k, v, j) \wedge \\ &\qquad ((i = \underline{2} \cdot j) \wedge (u = k \cdot k)) \vee ((i = \underline{2} \cdot j + 1) \wedge (u = \underline{2} \cdot k \cdot k)))))\end{aligned}$$

where we used the abbreviation

$$\begin{aligned}&\mathrm{EXPCOMPinit}(a, b) \equiv \\ &(\mathrm{BIT}'(a, \underline{8}) \wedge \neg \mathrm{BIT}'(a, \underline{4}) \wedge \mathrm{BIT}'(a, \underline{2}) \wedge \\ &\quad \mathrm{BIT}'(b, \underline{8}) \wedge \mathrm{BIT}'(b, \underline{4})) \vee \\ &(\mathrm{BIT}'(a, \underline{16}) \wedge \neg \mathrm{BIT}'(a, \underline{8}) \wedge \neg \mathrm{BIT}'(a, \underline{4}) \wedge \mathrm{BIT}'(a, \underline{2}) \wedge \\ &\quad \mathrm{BIT}'(b, \underline{16}) \wedge \neg \mathrm{BIT}'(b, \underline{8}) \wedge \neg \mathrm{BIT}'(b, \underline{4})) \vee \\ &(\mathrm{BIT}'(a, \underline{32}) \wedge \neg \mathrm{BIT}'(a, \underline{16}) \wedge \neg \mathrm{BIT}'(a, \underline{8}) \wedge \neg \mathrm{BIT}'(a, \underline{4}) \wedge \mathrm{BIT}'(a, \underline{2}) \wedge \\ &\quad \neg \mathrm{BIT}'(b, \underline{32}) \wedge \mathrm{BIT}'(b, \underline{16}) \wedge \neg \mathrm{BIT}'(b, \underline{8}) \wedge \mathrm{BIT}'(b, \underline{4}))\end{aligned}$$

Proposition 5.4.22. $\mathrm{EXPCOMP}(a,b) \in \Pi_0^B$

Proof. Immediately by inspection of Definition 5.4.21, using Propositions 5.4.12 and 5.4.15. □

Lemma 5.4.23. *If* $\mathbb{N} \models \mathrm{EXPCOMP}(\underline{n}, \underline{m})$ *and* i *and* j *are consecutive bits in* n*, then* i *consists of the bits* $i \ldots (j{+}1)$ *of* m*, that is,* $i = \sum_{\ell=0}^{i-(j+1)} m[\ell + j + 1] \cdot 2^\ell$.

Proof. Assume $\mathbb{N} \models \mathrm{EXPCOMP}(\underline{n}, \underline{m})$ and use Propositions 5.4.20, 5.4.16, and 5.4.13.

We argue by induction on i. If i and j are the smallest consecutive bits in n the claim follows by inspection of the base cases in $\mathrm{EXPCOMPinit}(\underline{n}, \underline{m})$, compare Figure 3.

So let i and j be consecutive bits in n, with j not being the smallest. Then there is a j' such that j, j' are consecutive bits in n. By induction hypothesis $j = \sum_{\ell=0}^{j-(j'+1)} m[\ell + j' + 1] \cdot 2^\ell$. From $\mathbb{N} \models \mathrm{EXPCOMP}(\underline{n}, \underline{m})$ we therefore get for $\tilde{\imath} = \sum_{\ell=0}^{i-(j+1)} m[\ell + j + 1] \cdot 2^\ell$ that $\tilde{\imath} = 2j \wedge 2^i = (2^j)^2$ or $\tilde{\imath} = 2j + 1 \wedge 2^i = 2(2^j)^2$. In either case $\tilde{\imath} = i$ follows. □

Lemma 5.4.24. *If* $n \geq 3$ *then there are* $a, b \leq 2{\cdot}2^n$ *with* $\mathbb{N} \models \mathrm{EXPCOMP}(\underline{a}, \underline{b})$ *such that* n *is the highest bit of* a *and there is a next bit* i *in* a *such that* $n = \sum_{\ell=0}^{n-(i+1)} b[\ell] \cdot 2^\ell$.

Proof. Let $n \geq 3$. We have to show that there are a, b and i such that $\mathbb{N} \models \mathrm{EXPCOMP}(\underline{a}, \underline{b})$ and n is the highest and i the second highest bit in a and $n = \sum_{\ell=0}^{n-(i+1)} b[\ell] \cdot 2^\ell$.

We argue by induction on n and tacitly use Propositions 5.4.20, 5.4.16, and 5.4.13.

The cases $n = 3$ and $n = 4$, and $n = 5$ are witnessed by $a = 10, b = 12, i = 1$ and $a = 18, b = 16, i = 1$ and $a = 34, b = 18, i = 1$, respectively.

So let $n \geq 6$. Then there is an $n' \geq 3$ such that $n = 2n'$ or $n = 2n'{+}1$. By induction hypothesis there are a', b', i' with $a'[n']$ and $\mathbb{N} \models \mathrm{EXPCOMP}(a', b')$ and n', i' the two highest bits of a and $n' = \sum_{\ell=0}^{n'-(i'+1)} b'[\ell] \cdot 2^\ell$. Then

$$a = 2^n + a' \qquad \text{and} \qquad b = n \cdot 2^{n'}$$

are as desired. □

Definition 5.4.25. We define the abbreviation

$$\begin{aligned}
&\mathrm{EXP}(x,y) \equiv \\
&(x = 0 \wedge y = 1) \vee (x = 1 \wedge y = \underline{2}) \vee (x = \underline{2} \wedge y = \underline{4}) \vee \\
&(x \geq \underline{3} \wedge (\exists a \leq \underline{2} \cdot y)(\exists b \leq \underline{2} \cdot y)(\exists u \leq y) \\
&\qquad (\mathrm{EXPCOMP}(a,b) \wedge \mathrm{CONSEC}(a,y,u) \wedge \mathrm{BITCOPY}(b,y,u,x)))
\end{aligned}$$

Proposition 5.4.26. $\mathrm{EXP}(a, b) \in \Pi^B_0$

Proof. Immediately by inspection of Definition 5.4.25, using Propositions 5.4.22, 5.4.19, and 5.4.15. □

Lemma 5.4.27. $\mathbb{N} \models \mathrm{EXP}(\underline{n}, \underline{m})$ *if and only if* $2^n = m$.

Proof. Immediately from Lemmata 5.4.23 and 5.4.24. □

Definition 5.4.28. We define $\mathrm{BIT}(x, i) \equiv \exists k \leq x(\mathrm{EXP}(i, k) \wedge \mathrm{BIT}'(x, k))$

Proposition 5.4.29. $\mathrm{BIT}(x, i) \in \Pi^B_0$

Proof. Immediately by inspection of Definition 5.4.28, using Proposition 5.4.26. □

Definition 5.4.30. We define the abbreviation

$$\begin{aligned}\mathrm{LOG}(x, i) \equiv\ & (i = 0 \wedge x = 0) \vee \\ & (\exists u \leq x)(\mathrm{EXP}(i, u) \wedge (\forall v \leq x)(v > u \rightarrow \neg \mathrm{POW}_2(v)))\end{aligned}$$

Proposition 5.4.31. $\mathrm{LOG}(b, a) \in \Pi^B_0$

Proof. Immediately by inspection of Definition 5.4.30, using Propositions 5.4.26 and 5.4.8. □

Proposition 5.4.32. $\mathbb{N} \models \mathrm{LOG}(\underline{m}, \underline{n})$ *if and only if* $m > 0$ *and* $n = \lfloor \log_2(m) \rfloor$ *or* $m = n = 0$.

Proof. Immediately from Proposition 5.4.9 and Lemma 5.4.27. □

Recall that, by Lemma 5.2.18, induction for $\Sigma^B_0(\alpha)$, that is, for all bounded first-order formulae, is available in $V^0(\alpha)$. Therefore, it is easy to show that the expected properties of the defined predicates are theorems of $V^0(\alpha)$.

In particular, as theorems of $V^0(\alpha)$, a number x is a power of two if and only $2x$ is, products of powers of two are again powers of two, for each x power of two, there is a $y \leq x$ power of two such that $x = y^2$ or $x = 2y^2$, and so on. Moreover, still as a theorem of $V^0(\alpha)$, two numbers are equal if and only if their binary representations, in the sense of $\mathrm{BIT}'(\cdot, \cdot)$, coincide. Using the rules of division and remainder, available in $V^0(\alpha)$ immediately from the definition, it is easy to see that numbers, seen as bit-strings, can be concatenated. Finally, $V^0(\alpha)$ knows that $\mathrm{LOG}(y, x)$ is the graph of a function, that is, for each x there is precisely one y such that $\mathrm{LOG}(y, x)$. This will justify Notation 5.5.1.

5.5 The Theories $V^{i/j}(\alpha)$

As mentioned in the beginning of this section (on page 63) one of the restrictions of bounded arithmetic is, that induction is only available up to $\log(x)$ and, given that exponentiation is not available, this is a proper restriction. It is therefore natural to consider theories with induction further restricted, allowing iteration only up to a (fixed) iterate of the logarithm function.

Notation 5.5.1. For s and t number terms we write $s = |t|$ as a shorthand for $\mathrm{LOG}(t, s)$.

Remark 5.5.2. Recall that in Proposition 5.4.31 we showed that $s = |t|$ is a Π^B_0-formula.

Definition 5.5.3. For $n \in \mathbb{N}$ a natural number, we define by (meta)induction on n the formula $s = |t|_n$ as follows

$$\begin{array}{lcl} s = |t|_0 & \equiv & s = t \\ s = |t|_{n+1} & \equiv & \exists x \leq t(x = |t| \wedge s = |x|_n) \end{array}$$

Proposition 5.5.4. $s = |t|_n \in \Pi^B_0$

Proof. By induction on n, using Remark 5.5.2. □

Definition 5.5.5 ($\mathcal{F}$-L^j-Induction Rule)**.** If $\mathcal{F}$ is a set of formulae, then by "$\mathcal{F}$-L^j-Induction" we denote the scheme

$$\frac{\Gamma, \neg A(a), A(a+1)}{\Gamma, \neg(s = |t|_j), \neg A(0), A(s)}$$

for all formulae $A(x) \in \mathcal{F}$, where a is required to be an eigenvariable, that is, is required not to be free in $\Gamma, A(0)$. Here s, t may be any terms of the language.

Definition 5.5.6 ($V^{i/j}(\alpha)$)**.** The system $V^{i/j}(\alpha)$ is defined to be $V^0(\alpha)$ plus $\Sigma^B_i(\alpha)$-L^j-Induction.

Remark 5.5.7. Note that the t in the $\mathcal{F}$-L^j-Induction Rule only serves to witness that the place up to which we do induction is small, compared to the size of usual number terms. In the special case $j = 0$ we can take $t = s$ and the alternative $s \neq s$ can be cut against the corresponding equality axiom. So, in essence, $V^{i/0}$ has the following Σ^B_i-induction rule

$$\frac{\Gamma, \neg A(a), A(a+1)}{\Gamma, \neg A(0), A(s)}$$

Proposition 5.5.8. *Over V^0, the $\Sigma_i^B(\alpha)$-L^j-induction rule implies the $\Pi_i^B(\alpha)$-L^j-induction rule.*

Proof. Argue informally in the system with V^0 and $\Sigma_i^B(\alpha)$-L^j-induction.

Let $B(x)$ be Π_i^B and assume $\forall a(\neg B(a) \vee B(a+1))$. Moreover, let N be short for $|t|_j$. We have to show $\neg B(0) \vee B(N)$.

Let x be arbitrary and instantiate our assumption by $N - x - 1$. This yields $\neg B(N-(x+1)) \vee B(N-x)$. Since x was arbitrary we have $\forall x[B(N-x) \vee \neg B(N-(x+1))]$. Using the abbreviation $C(x) \equiv \neg B(N-x)$ this can be written as $\forall x[\neg C(x) \vee C(x+1)]$. Hence induction on the $\Sigma_i^B(\alpha)$-formula $C(x)$ yields $\neg C(0) \vee C(N)$ which is the same as $B(N) \vee \neg B(0)$. This finishes the proof. □

6 Propositional Translations

Having a boundedness result available with Theorem 4.2.17, we can obtain limits on the provability of the sequential iteration principle for certain arithmetical theories—provided we can translate them into AC^0-Tait in a shallow way. This translation is carried out in this section.

Propositional translations have a long tradition in mathematical logic. In the context of Bounded Arithmetic they are often called "Parris-Wilkie translation" due to their use by Paris and Wilkie [50]. Already Schütte's ω-rule [56] essentially is a propositional translation: a statement $\forall x A(x)$ is shown by showing $A(\underline{n})$ for every $n \in \mathbb{N}$, as if $\forall x A(x)$ was a big conjunction. In the Tait calculus, propositional translations were used already in the very paper [62] that introduced this calculus.

6.1 Translating Formulae

The first step in our propositional translation is to associate a propositional formula to every formula in our language $\mathcal{L}_2(\alpha)$ of two-sorted arithmetic. The translation will be guided by the intuition that the string sort corresponds to finite sequences of truth values; the number sort will be translated away, by unfolding bounded number quantifiers. However, the translation of a string $\mathfrak{X}$ as bit-vectors has an underlying size parameter $|\mathfrak{X}|$. So we will end up with dynamic formulae.

Notation 6.1.1. To have the presentation more uniform we use $\forall_0 A$ and $\exists_0 A$ as abbreviations for A. Similarly, $\forall_{-1} A$ and $\exists_{-1} A$ are also abbreviations for A.

Also, we use $\bigvee_0$, $\bigwedge_0$, $\bigvee_1 A$, and $\bigwedge_1 A$ as abbreviations for F, T, A, and A, respectively.

Definition 6.1.2 ($\|A\|_{\vec{n}}$)**.** By induction on $\mathcal{L}_2(\alpha)$-formulae we define for every Π^B_∞-formula A *without free number variables*, every list $\mathfrak{X}_1, \ldots, \mathfrak{X}_k$ of second order variables such that the free variables of A are among the $\mathfrak{X}_1, \ldots, \mathfrak{X}_k$, and every list $n_1, \ldots, n_k$ of natural numbers a propositional formula (Definition 3.1.4)

$$\|A(\vec{\mathfrak{X}})\|_{\vec{n}}$$

as follows.

- For the "arithmetic" atomic formulae we set

$$\|s(|\vec{\mathfrak{X}}|) = t(|\vec{\mathfrak{X}}|)\|_{\vec{n}} = \begin{cases} \mathrm{T} & (s(\underline{\vec{n}}))^{\mathbb{N}} = (t(\underline{\vec{n}}))^{\mathbb{N}} \\ \\ \mathrm{F} & \text{otherwise} \end{cases}$$

and similar for $s \neq t$, $s \leq t$ and $s \geq t$.

- For atomic formulae $\mathfrak{X}_i{}^{[\vec{s}(|\vec{\mathfrak{X}}|)]}(t(|\vec{X}|))$ we set

$$\|\mathfrak{X}_i{}^{[\vec{s}]}(t(|\vec{X}|))\|_{\vec{n}} = \begin{cases} p_j^{\mathfrak{X}_i} & j < n_i - 1 \\ \mathrm{T} & j = n_i - 1 \\ \mathrm{F} & \text{otherwise} \end{cases}$$

where

$$j = (\langle s_1(\underline{\vec{n}}); \langle \ldots; \langle s_k(\underline{\vec{n}}); t(\underline{\vec{n}})\rangle\rangle\rangle)^{\mathbb{N}}$$

The definition for negated string terms $\bar{\mathfrak{X}}_i^{[\vec{s}(|\vec{X}|)]}(t(|\vec{X}|))$ is similar.

- For parameter formulae $\alpha(\mathfrak{X}_i{}^{[\vec{s}(|\vec{X}|)]})$ we set

$$\|\alpha(\mathfrak{X}_i{}^{[\vec{s}]})\|_{\vec{n}} = \bigvee_{\ell \leq n_i} (\, (\bigwedge_{\ell \leq j < n_i} \neg \|\mathfrak{X}_i{}^{[\vec{s}]}(\underline{j})\|_{\vec{n}}) \wedge \|\mathfrak{X}_i{}^{[\vec{s}]}(\underline{\ell - 1})\|_{\vec{n}}$$

$$\wedge \alpha_{\ell-1}(\|\mathfrak{X}_i{}^{[\vec{s}]}(\underline{\ell-2})\|_{\vec{n}}, \ldots, \|\mathfrak{X}_i{}^{[\vec{s}]}(\underline{0})\|_{\vec{n}}) \,)$$

and similar for the other cases (negated parameter, negated variable). Here the understanding is that the conjuncts $\alpha_{-1}()$ and $\|\mathfrak{X}_i{}^{[\vec{s}]}(\underline{-1})\|_{\vec{n}}$ should be considered as not present.

- For conjunction and disjunction $\|\cdot\|_{\vec{n}}$ is homomorphic, that is $\|A \wedge B\|_{\vec{n}} = \|A\|_{\vec{n}} \wedge \|B\|_{\vec{n}}$ and $\|A \vee B\|_{\vec{n}} = \|A\|_{\vec{n}} \vee \|B\|_{\vec{n}}$.

- Bounded number quantification is translated as big disjunctions and conjuctions.

$$\|\forall x < t(|\vec{\mathfrak{X}}|)\, A(x, \vec{\mathfrak{X}})\|_{\vec{n}} \;=\; \bigwedge_{i < (t(\underline{\vec{n}}))^{\mathbb{N}}} \|A(\underline{i}, \vec{\mathfrak{X}})\|_{\vec{n}}$$

$$\|\exists x < t(|\vec{\mathfrak{X}}|)\, A(x, \vec{\mathfrak{X}})\|_{\vec{n}} \;=\; \bigvee_{i < (t(\underline{\vec{n}}))^{\mathbb{N}}} \|A(\underline{i}, \vec{\mathfrak{X}})\|_{\vec{n}}$$

- Bounded string quantification is translated by (administrative) boolean quantification.

$$\|\forall \mathfrak{X} < t(|\vec{\mathfrak{X}}|)\, A(\mathfrak{X}, \vec{\mathfrak{X}})\|_{\vec{n}} \;=\; \bigwedge_{i < (t(\underline{\vec{n}}))^{\mathbb{N}}} \forall_{i-1} p_0^{\mathfrak{X}} \ldots p_{i-2}^{\mathfrak{X}} \|A(\mathfrak{X}, \vec{\mathfrak{X}})\|_{i,\vec{n}}$$

$$\|\exists \mathfrak{X} < t(|\vec{\mathfrak{X}}|)\, A(\mathfrak{X}, \vec{\mathfrak{X}})\|_{\vec{n}} \;=\; \bigvee_{i < (t(\underline{\vec{n}}))^{\mathbb{N}}} \exists_{i-1} p_0^{\mathfrak{X}} \ldots p_{i-2}^{\mathfrak{X}} \|A(\mathfrak{X}, \vec{\mathfrak{X}})\|_{i,\vec{n}}$$

Remark 6.1.3. $\|\cdot\|_{\vec{n}}$ is homomorphic with respect to negation, that is $\|\neg A\|_{\vec{n}} \equiv \neg\|A\|_{\vec{n}}$.

Proof. Induction on A. □

Proposition 6.1.4. $\mathsf{dp}(\|A\|_{\vec{n}}) \leq 2\mathsf{dp}(A) + 1$

Proof. Trivial induction on A.

(The factor 2 comes in, since the translation of $\exists_{<}\mathfrak{X}t$ or $\forall_{<}\mathfrak{X}t$ is of the form $\bigvee_k \exists_k$ or $\bigwedge_k \forall_k$, respectively. The additional constant is due to the fact that $\mathsf{dp}(\|\alpha(\mathfrak{T})\|_{\vec{n}}) = 3$.) □

Definition 6.1.5 ($[\![A]\!]$)**.** If A is a $\mathcal{L}_2(\alpha)$-formula in Π^B_∞ we define its *propositional translation* $[\![A]\!]$ to be the dynamic formula

$$[\![A]\!] = (\|A(\underline{n}, \ldots, \underline{n}, \mathfrak{X}_1, \ldots, \mathfrak{X}_k)\|_{n,\ldots,n})_{n\in\mathbb{N}}$$

where $\vec{x}$ is an enumeration of the free number variables and $\mathfrak{X}_1, \ldots, \mathfrak{X}_k$ is an enumeration of the free string variables of $A(\vec{x}, \vec{\mathfrak{X}})$.

Remark 6.1.6. Immediately from the symmetry of Definition 6.1.2 we see that $[\![A]\!]$ is well defined, i.e., is independent of the order in which the free string and number variables are enumerated.

Proposition 6.1.7. *If $A \in \Sigma^B_i(\alpha)$ then $[\![A]\!] \in \Sigma^q_i(\alpha)$.*

Proof. Induction on A, following the inductive Definition 5.1.17 of $\Sigma^B_i(\alpha)$. □

Lemma 6.1.8. *If $t(\vec{x}, |\vec{X}|)$ is a term of the language $\mathcal{L}_2(\alpha)$ of two-sorted bounded arithmetic, $\vec{\mathfrak{p}}, \vec{\mathfrak{q}}$ are polynomials, then the function*

$$\begin{array}{rcl} \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & \left(t(\underline{\overrightarrow{\mathfrak{p}(n)}}, \underline{\overrightarrow{\mathfrak{q}(n)}})\right)^{\mathbb{N}} \end{array}$$

is a polynomial function.

Proof. By induction on the buildup of t, see Definition 5.1.1. □

Lemma 6.1.9. *If $A(\vec{x}, \vec{X})$ is a $\mathcal{L}_2(\alpha)$-formula, $\vec{\mathfrak{p}}, \vec{\mathfrak{q}}$ are polynomial functions, there is a polynomial function $\mathfrak{f}$ such that for every $n \in \mathbb{N}$ and all $\vec{k} \leq \vec{\mathfrak{p}}(n)$ and all $\vec{\ell} \leq \vec{\mathfrak{q}}(n)$ we have*

$$|\|A(\underline{\vec{k}}, \vec{X})\|_{\vec{\ell}}| \leq \mathfrak{f}(n)$$

Proof. Induction on A. □

Corollary 6.1.10. $|[\![A]\!]| \in \mathfrak{n}^{\mathcal{O}(1)}$

Proof. Lemma 6.1.8 with all the $\vec{\mathfrak{p}}, \vec{\mathfrak{q}}$ the identity, and Lemma 6.1.9. □

6.2 Translating $V^0(\alpha)$ Proofs

Definition 6.2.1. We write $\vdash^h_{\star,c} \Gamma$ to denote $\emptyset \vdash^h_{w,c} \Gamma$ where $w = \max\{|A| \mid A \in \Gamma\}$.

Proposition 6.2.2. *If purely propositional, α-free, closed formula A is true, then $\vdash^{\mathsf{dp}(A)}_{\star,0} A$.*

Proof. By Induction on A.

The only true closed atom is T, which can be derived in a derivation of height 1.

If A is true and is of the form $\bigvee_k \vec{A}$ then, for some $1 \leq i \leq k$ we have that A_i is true. Therefore we have $\vdash^{\mathsf{dp}(A_i)}_{\star,0} A_i$ and hence, by one $\bigvee_k$-rule we have $\vdash^{\mathsf{dp}(A_i)+1}_{\star,0} \bigvee_k \vec{A}$.

If A is true and of the form $\bigwedge_k \vec{A}$ then for all $1 \leq i \leq k$ we have that A_i is true and therefore, by induction hypothesis, $\vdash^{\mathsf{dp}(A_i)}_{\star,0} A_i$. So, by one $\bigwedge_k$-rule we have $\vdash^{\max\{\mathsf{dp}(A_i)|1\leq i\leq k\}+1}_{\star,0} \bigwedge_k \vec{A}$. □

Lemma 6.2.3. *If $A(\vec{x}, |\vec{\mathfrak{X}}|)$ is a true bounded first-order formula without atoms of the form $\mathfrak{T}(t)$ or $\alpha(\mathfrak{T})$, then $\vdash^{2\cdot\mathsf{dp}(A)+1}_{\star,0} \|A(\underline{\vec{m}}, |\vec{\mathfrak{X}}|)\|_{\vec{n}}$ for* arbitrary *$\vec{m}, \vec{n} \in \mathbb{N}$. Here $w = |\|A(\underline{\vec{m}}, |\vec{\mathfrak{X}}|)\|_{\vec{n}}|$.*

Proof. By Proposition 6.1.4 the depth $\mathsf{dp}(\|A\|_{\vec{n}})$ is bound by the constant $2 \cdot \mathsf{dp}(A) + 1$, so the claim follows from Proposition 6.2.2.

Note that by induction on A we can show that $\|A(\underline{\vec{m}}, |\vec{\mathfrak{X}}|)\|_{\vec{n}}$ is true, if $A(\underline{\vec{m}}, \underline{\vec{n}})$ is. □

Proposition 6.2.4. *There is a constant depth proof of the propositional translation of the row axiom.*

Proof. We note that $\|\mathfrak{X}^{[\vec{t}]}(s) \leftrightarrow \mathfrak{X}(\langle t_1; \langle \ldots; \langle t_k; s\rangle\rangle\rangle)\|_{\vec{n}} \equiv \wp \leftrightarrow \wp$ for some $\wp \in \{\mathrm{T}, \mathrm{F}, p^{\mathfrak{X}}_0, p^{\mathfrak{X}}_1, p^{\mathfrak{X}}_2, \ldots\}$ which is a constant-depth logical tautology. Hence the claim follows by Proposition 6.2.2. □

Lemma 6.2.5. *If $A(\vec{x}, \vec{\mathfrak{X}})$ is one of the basic axioms (Definition 5.2.3), then, for some constant $c \in \mathbb{N}$, we have*

$$\vdash^c_{\star,0} \|A(\underline{\vec{n}}, \vec{\mathfrak{X}})\|_{\vec{n'}}$$

for all $\vec{n}, \vec{n'} \in \mathbb{N}$.

Proof. All the first order basic axioms are true formulae with an upper bound on their depths and size. Hence the claim follows from Lemma 6.2.3.

So the only basic axioms we still have to look at are

- $\mathfrak{X}(y) \to y < |\mathfrak{X}|$,
- $y + 1 = |\mathfrak{X}| \to \mathfrak{X}(y)$, and
- $t = t' \to \mathfrak{X}(t) \to \mathfrak{X}(t')$.

Let n and $\vec{m}$ be natural numbers and abbreviate $(t(\underline{\vec{m}}))^{\mathbb{N}}$ by i. Then

$$\|\mathfrak{X}(t(\underline{\vec{m}})) \to t(\underline{\vec{m}}) < |\mathfrak{X}|\|_n = \begin{cases} p_i^{\mathfrak{X}} \to \mathrm{T} & i < n-1 \\ \mathrm{T} \to \mathrm{T} & i = n-1 \\ \mathrm{F} \to \mathrm{F} & \text{otherwise} \end{cases}$$

and therefore can be derived using the truth axiom, followed by an $\vee$-introduction.

Moreover, with the same notations,

$$\|t(\underline{\vec{m}}) + 1 = |\mathfrak{X}| \to \mathfrak{X}(t(\underline{\vec{m}}))\|_n = \begin{cases} \mathrm{F} \to p_i^{\mathfrak{X}} & i < n-1 \\ \mathrm{T} \to \mathrm{T} & i = n-1 \\ \mathrm{F} \to \mathrm{F} & \text{otherwise} \end{cases}$$

and again, this can be derived using the truth axiom, followed by an $\vee$-introduction.

Finally, let $\vec{m}, n \in \mathbb{N}$. We use the abbreviations $i = (t(\underline{\vec{m}}))^{\mathbb{N}}$ and $j = (t'(\underline{\vec{m}}))^{\mathbb{N}}$. Then the propositional translation of the string extensionality axiom is of the following shape.

$$\|t(\underline{\vec{m}}) = t'(\underline{\vec{m}}) \to \mathfrak{X}(t(\underline{\vec{m}})) \to \mathfrak{X}(t'(\underline{\vec{m}}))\|_n = \begin{cases} \mathrm{F} \to \ldots & i \neq j \\ \mathrm{T} \to p_i^{\mathfrak{X}} \to p_i^{\mathfrak{X}} & i = j < n-1 \\ \mathrm{T} \to \mathrm{T} \to \mathrm{T} & i = j = n-1 \\ \mathrm{T} \to \mathrm{F} \to \mathrm{F} & i = j > n-1 \end{cases}$$

In either case using the truth axiom or the identity axiom does the job, followed by two $\vee$-introductions. □

Lemma 6.2.6. *For some constants $c, c' \in \mathbb{N}$ we have*

$$\vdash^c_{\star, m^{c'}} \|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]} \to \alpha(\mathfrak{X}^{[\vec{s}]}) \to \alpha(\mathfrak{Y}^{[\vec{t}]})\|_{n,m,\vec{n}}$$

Proof. We use the abbreviations

$$k_i \;=\; (\langle s_1; \langle \ldots; \langle s_{n'}; i\rangle\rangle\rangle)^{\mathbb{N}}$$

$$\ell_i \;=\; (\langle t_1; \langle \ldots; \langle t_{m'}; i\rangle\rangle\rangle)^{\mathbb{N}}$$

in order to facilitate notation. Also, by abuse of notation, we write $p_j^{\mathfrak{X}}$ to stand for T or F, if $j = n-1$ or $j > n-1$, respectively. Similar for $\mathfrak{Y}$ and the negated variables.

For every i we have a derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{X}}_{k_i} \qquad p^{\mathfrak{Y}}_{\ell_i}, \neg p^{\mathfrak{Y}}_{\ell_i}}{\neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{Y}}_{\ell_i}, p^{\mathfrak{X}}_{k_i} \wedge \neg p^{\mathfrak{Y}}_{\ell_i}}\wedge}{\neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{Y}}_{\ell_i}, p^{\mathfrak{X}}_{k_i} \not\leftrightarrow \neg p^{\mathfrak{Y}}_{\ell_i}}\vee}{\neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{Y}}_{\ell_i}, \neg\|\forall i < |\mathfrak{X}|.\mathfrak{X}^{[\vec{s}]}(i) \leftrightarrow \mathfrak{Y}^{[\vec{t}]}(i)\|_{n,m,\vec{n}}}\bigvee_n}{\neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{Y}}_{\ell_i}, \neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,m,\vec{n}}}\vee$$

and similar derivations of $p^{\mathfrak{X}}_{k_i}, \neg p^{\mathfrak{Y}}_{\ell_i}, \neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,m,\vec{n}}$.

Writing $\overrightarrow{p^{\mathfrak{X}}_{\cdot}}$ for $p^{\mathfrak{X}}_{k_{j-2}}, \ldots, p^{\mathfrak{X}}_{k_0}$ and $\overrightarrow{p^{\mathfrak{Y}}_{\cdot}}$ for $p^{\mathfrak{Y}}_{\ell_{j-2}}, \ldots, p^{\mathfrak{Y}}_{\ell_0}$ we first get, for every i, a derivation

$$\dfrac{\dfrac{\dfrac{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, \neg p^{\mathfrak{X}}_{k_i}, p^{\mathfrak{Y}}_{\ell_i}}{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, \neg p^{\mathfrak{X}}_{k_i} \vee p^{\mathfrak{Y}}_{\ell_i}}\vee,\vee \qquad \dfrac{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, p^{\mathfrak{X}}_{k_i}, \neg p^{\mathfrak{Y}}_{\ell_i}}{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, \neg p^{\mathfrak{Y}}_{\ell_i} \vee p^{\mathfrak{X}}_{k_i}}\vee,\vee}{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, p^{\mathfrak{X}}_{k_i} \leftrightarrow p^{\mathfrak{Y}}_{\ell_i}}\wedge$$

and hence, by the parameter extensionality rule, we get the following derivation.

$$\dfrac{\bar\alpha_{n-1}(\overrightarrow{p^{\mathfrak{X}}_{\cdot}}), \alpha_{n-1}(\overrightarrow{p^{\mathfrak{X}}_{\cdot}}) \qquad \neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, p^{\mathfrak{X}}_{k_i} \leftrightarrow p^{\mathfrak{Y}}_{\ell_i}}{\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,n,\vec{n}}, \bar\alpha_{n-1}(\overrightarrow{p^{\mathfrak{X}}_{\cdot}}), \alpha_{n-1}(\overrightarrow{p^{\mathfrak{Y}}_{\cdot}})}\ (\text{all } i)$$

Now let $0 \le j \le m$. We set Δ_j to be the set $\{\neg p^{\mathfrak{Y}}_{\ell_{j-1}}, p^{\mathfrak{Y}}_{\ell_j}, p^{\mathfrak{Y}}_{\ell_{j+1}}, \ldots, p^{\mathfrak{Y}}_{\ell_n}\}$, again with the understanding that $\neg p^{\mathfrak{Y}}_{k_{-1}}$ should be considered as not present. Note that the empty sequent can be derived from the Δ_j by cuts only.

Adding to the above derivation an $\bigwedge$-rule to all the identity axioms for the elements of Δ_j, followed by an $\bigvee$-introduction we obtain a derivation of

$$\Delta_j, \neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,m,\vec{n}}, \bar\alpha_{n-1}(\overrightarrow{p^{\mathfrak{X}}_{\cdot}}), \|\alpha(\mathfrak{Y}^{[\vec{t}]})\|_{n,m,\vec{n}}$$

From here an $\bigvee$ rule gives one of the disjuncts of $\|\alpha(\mathfrak{X}^{[\vec{s}]})\|_{n,m,\vec{n}}$. The other can, in the context Δ_j, $\neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,m,\vec{n}}$, be trivially derived. Hence an $\bigwedge$-rule gives now

$$\Delta_j, \neg\|\mathfrak{X}^{[\vec{s}]} = \mathfrak{Y}^{[\vec{t}]}\|_{n,m,\vec{n}}, \neg\|\alpha(\mathfrak{X}^{[\vec{s}]})\|_{n,m,\vec{n}}\|\alpha(\mathfrak{Y}^{[\vec{t}]})\|_{n,m,\vec{n}} \,.$$

Applying the multi-cut rule to all these derivations removes the Δ_j and two $\vee$-rules finish the desired derivation. □

Lemma 6.2.7. *Let A be a quantified propositional formula. Then $\vdash^{\mathcal{O}(A)}_{\star,0} \neg A, (\mathrm{T} \leftrightarrow A)$ and $\vdash^{\mathcal{O}(A)}_{\star,0} A, (\mathrm{F} \leftrightarrow A)$.*

Proof. Consider the derivations

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Lemma 3.3.11}}{A, \neg A}}{\neg A, (\mathrm{T} \to A)} \qquad \dfrac{}{\neg A, \mathrm{T}}}{(A \to \mathrm{T})}}{\neg A, (\mathrm{T} \leftrightarrow A)}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Lemma 3.3.11}}{A, \neg A}}{A, (A \to \mathrm{F})} \qquad \dfrac{}{\neg A, \mathrm{T}}}{(\mathrm{F} \to A)}}{A, (\mathrm{F} \leftrightarrow A)}$$

which are as desired. □

Lemma 6.2.8. *Let $A(x, \vec{x}, \vec{\mathfrak{X}})$ be a $\Sigma^B_0(\alpha)$-formula, that is, a purely arithmetic formula. Then there is a constant $c \in \mathbb{N}$, depending only (in fact linearly) on $\mathsf{dp}(A)$, such that for all natural numbers $g, \vec{f}$ we have*

$$\vdash^c_{\star, g^2+1} \|\exists \mathfrak{Z} \leq \underline{g} \forall i < \underline{g}(\mathfrak{Z}(i) \leftrightarrow A(i, \underline{\vec{g}}, \vec{\mathfrak{X}}))\|_{\vec{f}} \,.$$

Proof. We use the abbreviation $\varphi_i \equiv \|A(\underline{i}, \underline{\vec{g}}, \vec{X})\|_{\vec{f}}$. Moreover, for $1 \leq f \leq g$ we set

$$\Delta_f = \{\neg\varphi_{f-1}, \varphi_f, \varphi_{f+1}, \ldots, \varphi_{g-1}\}$$

and we also set

$$\Delta_0 = \{\varphi_0, \ldots, \varphi_{g-1}\} \,.$$

It should be noted that all the Δ_i contain purely propositional formulae only and that from $\Delta_0, \ldots, \Delta_g$ the empty sequent can be derived using only cuts as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Delta_0 \qquad \Delta_1}{\varphi_1, \ldots, \varphi_{g-1}} \qquad \Delta_2}{\varphi_2, \ldots, \varphi_{g-1}} \qquad \Delta_3}{\cdots \qquad\qquad \cdots}}{\varphi_{g-1}} \qquad \Delta_g}{\emptyset}$$

We write Γ_f as a shorthand for the set

$$\Gamma_f = \{\neg(p_0 \leftrightarrow \varphi_0), \ldots, \neg(p_{f-2} \leftrightarrow \varphi_{f-2})\}$$

that will later serve as the comprehension context. Note that Γ_f contains $p_0, \ldots, p_{f-2}$ as free variables. For $0 \leq f \leq g$ we now can have the following derivations

$$\dfrac{\cdots \dfrac{\text{Proposition 3.3.11}}{(p_i \leftrightarrow \varphi_i), \neg(p_i \leftrightarrow \varphi_i)} \cdots \quad \dfrac{\text{Lemma 6.2.7}}{(\mathrm{T} \leftrightarrow \varphi_{f-1}), \neg\varphi_{f-1}} \quad \cdots \dfrac{\text{Lemma 6.2.7}}{(\mathrm{F} \leftrightarrow \varphi_i), \varphi_i} \cdots}{\dfrac{\dfrac{\dfrac{\dfrac{\|\forall i < \underline{g}(\mathfrak{Z}(i) \leftrightarrow A(i, \underline{\vec{g}}, \vec{\mathfrak{X}}))\|_{f,\vec{f}}, \Gamma_f, \Delta_f}{\exists_{f-1} p_0 \ldots p_{f-2} \|\forall i < \underline{g}(\mathfrak{Z}(i) \leftrightarrow A(i, \underline{\vec{g}}, \vec{\mathfrak{X}}))\|_{f,\vec{f}}, \Gamma_f, \Delta_f} \exists_{f-1}}{\|\exists \mathfrak{Z} < \underline{g} + 1 \forall i < \underline{g}(\mathfrak{Z}(i) \leftrightarrow A(i, \underline{\vec{g}}, \vec{\mathfrak{X}}))\|_{\vec{f}}, \Gamma_f, \Delta_f} \bigvee_g}{\|\exists \mathfrak{Z} < \underline{g} + 1 \forall i < \underline{g}(\mathfrak{Z}(i) \leftrightarrow A(i, \underline{\vec{g}}, \vec{\mathfrak{X}}))\|_{\vec{f}}, \Delta_f} \text{comp}} \bigwedge_g$$

Taking all these derivations and applying one multi-cut rule yields the claim. □

Theorem 6.2.9. *For every $V^0(\alpha)$-proof of Γ with only $\Sigma^B_1(\alpha)$ cuts there are polynomials $\mathfrak{p}, \mathfrak{q}$, and there is a constant $c \in \mathbb{N}$ such that for all $\vec{f}, \vec{g} \in \mathbb{N}$*

$$\vdash^c_{\mathfrak{p}(\vec{f},\vec{g}),\mathfrak{q}(\vec{f},\vec{g})} \|\Gamma(\underline{\vec{g}})\|_{\vec{f}}$$

Proof. Induction on the derivation. If the derivation is a basic axiom, then the claim follows from Lemma 6.2.5. If it is the string extensionality axiom, the claim follows from Lemma 6.2.6. If it is the row axiom, the claim follows by Proposition 6.2.4.

If the derivation is a $\Sigma^B_0(\alpha)$-comprehension axiom, then the claim follows from Lemma 6.2.8 using that by Lemma 6.1.9 the formulae are of small size.

If the last inference is a cut, the cut formula necessarily is $\Sigma^q_1(\alpha)$-formula (compare Proposition 6.1.7), that is, is in the Σ-closure of the purely propositional formulae. Therefore, we can refer to Corollary 3.4.7. Note that the sum of two constants still is a constant.

So, what remains are the rules of two-sorted predicate logic. The propositional rules can be translated into the corresponding propositional rules on the translated formulae. Since Γ consists only of bounded formulae, the only thing still to be considered are the introduction rules for the bounded quantifiers.

If the last rule was an $\forall x < t$-introduction

$$\frac{\Gamma, a \geq t, A(a, \vec{x})}{\Gamma, \forall x < t.A(x, \vec{x})}$$

the induction hypotheses give us derivations of $\|\Gamma(\vec{g})\|_{\vec{f}}, \mathrm{F}, \|A(\underline{i}, \vec{g})\|_{\vec{f}}$ for $0 \leq i < t^{\mathbb{N}}$. By strengthening we can get rid of the F and an $\bigwedge_{t^{\mathbb{N}}}$ introduction gives the desired derivation.

If the last rule was

$$\frac{\Gamma, s(\vec{x}, |\vec{\mathfrak{X}}|) < t(\vec{x}, |\vec{\mathfrak{X}}|) \wedge A(s, \vec{x})}{\Gamma, \exists x < t.A(x, \vec{x})}$$

we argue similarly, using $\wedge$-inversion and $\bigvee_{t^{\mathbb{N}}}$-introduction.

If the last rule was

$$\frac{\Gamma, |\mathfrak{Y}| \geq t(\vec{x}, |\vec{\mathfrak{X}}|), A(\vec{x}, \mathfrak{Y}, \vec{\mathfrak{X}})}{\Gamma, \forall \mathfrak{X} < t.A(\vec{x}, \mathfrak{Y}, \vec{X})}$$

we argue as follows. Let $\mathfrak{p}'(\vec{f}, \vec{g})$ be a polynomial bounding $(t(\underline{\vec{f}}, \underline{\vec{g}}))^{\mathbb{N}}$. By induction hypothesis, derivations of $\|\Gamma(\underline{\vec{g}})\|_{\vec{f}}, \mathrm{F}, \|A(\underline{\vec{g}}, \mathfrak{Y}, \vec{\mathfrak{X}})\|_{f, \vec{f}}$ for all $f < t^{\mathbb{N}} \leq \mathfrak{p}'(\vec{f}, \vec{g})$. The eigenvariable condition guarantees that the $p_0^{\mathfrak{Y}}, \ldots, p_{f-2}^{\mathfrak{Y}}$ are eigenvariables. So by strengthening and $\forall_{f-1}$-introduction we get derivations of $\|\Gamma(\underline{\vec{g}})\|_{\vec{f}}, \forall_{f-1} p_0^{\mathfrak{Y}} \ldots p_{f-2}^{\mathfrak{Y}} \|A(\underline{\vec{g}}, \mathfrak{Y}, \vec{\mathfrak{X}})\|_{f, \vec{f}}$ for all $f < t^{\mathbb{N}}$. An $\bigwedge_{t^{\mathbb{N}}}$-introduction finishes the derivation.

If the last rule was

$$\frac{\Gamma, |\mathfrak{Y}| < t(\vec{x}, |\vec{\mathfrak{X}}|) \wedge A(\vec{x}, \mathfrak{Y}, \vec{\mathfrak{X}})}{\Gamma, \exists \mathfrak{Y} < t.A(\vec{x}, \mathfrak{Y}, \vec{\mathfrak{X}})}$$

We distinguish between $f < t(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}}$ and $f \geq t^{\mathbb{N}}$. In the second case, the induction hypothesis yields $\|\Gamma(\underline{\vec{g}})\|_{\vec{f}}, \mathrm{F} \wedge \ldots$, so $\wedge$-inversion and strengthening yield the claim.

In the first case the induction hypothesis yields us a proof of $\|\Gamma(\underline{\vec{g}})\|_{f, \vec{f}}, \mathrm{T} \wedge \|A(\underline{\vec{g}}, \mathfrak{Y}, \vec{\mathfrak{X}})\|_{f, \vec{f}}$ and $\wedge$-inversion followed by $\exists_{f-1}$-introduction and $\bigvee_{t^{\mathbb{N}}}$-introduction yields the claim. □

Corollary 6.2.10. *For every V^0-proof of Γ with only Σ_1^B cuts, in particular for every free-cut free proof, there is a dynamic proof $\mathfrak{d}$ such that*

$$\mathfrak{d} \vdash^{\mathcal{O}(1)}_{\mathfrak{n}^{\mathcal{O}(1)}, \mathfrak{n}^{\mathcal{O}(1)}} [\![\Gamma]\!]$$

Proof. Use Theorem 6.2.9. □

6.3 Translating Induction

As is well known in proof theory, induction up to each particular number n can be translated into a series of cuts. In other words, from $A(0) \to A(1)$ and $A(1) \to A(2)$ we can conclude—using only propositional logic—that $A(0) \to A(2)$; together with $A(2) \to A(3)$ this implies $A(0) \to A(3)$ and so on.

To obtain the optimal height, we do not cut in the linear way just sketched. Instead we use a balanced tree of cuts, that is, we obtain a proof of $A(0) \to A(n)$ by cutting proofs of $A(0) \to A(\lfloor n/2 \rfloor)$ and $A(\lfloor n/2 \rfloor) \to A(n)$. In this way, our proofs will have only logarithmic height.

Lemma 6.3.1. *If n' is a power of 2, $1 \leq n \leq n'$, $\mathcal{C}$ is a set of formulae, $A_1, \ldots, A_n \in \mathcal{C}$, and for all $0 \leq i < n$ we have*

$$\vdash^{h}_{\mathcal{C};w,c} \Gamma, \neg A_i, A_{i+1}$$

then

$$\vdash^{h+\log(n')}_{\mathcal{C};w,c} \Gamma, \neg A_0, A_n$$

Proof. By induction on n'. If $n' = 1$, then there is nothing to show as the claim is already one of the premises.

So let $n' = 2m$. We may without loss of generality assume that $n > m$ for otherwise the claim is directly the induction hypothesis. Now, by induction hypothesis get

$$\vdash^{h+\log(m)}_{\mathcal{C};w,c} \Gamma, \neg A_0, A_m \qquad \text{and} \qquad \vdash^{h+\log(m)}_{\mathcal{C};w,c} \Gamma, \neg A_m, A_n$$

By an additional cut on $A_m \in \mathcal{C}$ we get the claim. □

Remark 6.3.2. In Lemma 6.3.1 we can always choose n' to be $n' = 2^{\log(n)}$ which is a power of 2, and by Proposition 2.1.24 we have $n \leq 2^{\log(n)} = n'$. Moreover, $\log(n') = \log\left(2^{\log(n)}\right) = \log(n)$ by Proposition 2.1.26. So, in Lemma 6.3.1 we can read the $\log(n')$ as $\log(n)$.

Recall Definition 2.1.10. In particular, we will use $\log^{(j+1)}(n)$ as an abbreviation for $\underbrace{\log(\log(\ldots(\log(}_{j+1} n))))$.

Theorem 6.3.3. *For every $V^{i/j}(\alpha)$-proof of Γ with only $\Sigma^B_{i+1}(\alpha)$ cuts there are polynomials $\mathfrak{p}, \mathfrak{q}, \mathfrak{r}$, and there are constants $c, C \in \mathbb{N}$ such that for all $\vec{f}, \vec{g} \in \mathbb{N}$*

$$\vdash^{c\log^{(j+1)}(\mathfrak{r}(\vec{f},\vec{g}))+C}_{\Sigma^q_i(\alpha);\mathfrak{p}(\vec{f},\vec{g}),\mathfrak{q}(\vec{f},\vec{g})} \left\|\Gamma(\underline{\vec{g}})\right\|_{\vec{f}}$$

Proof. We argue as in the proof of Theorem 6.2.9. As for cuts, note that for any two elements of the set $\{c \cdot \log^{(j+1)}(\mathfrak{r}) + C \mid c, C \in \mathbb{N},\ \mathfrak{r} \text{ a polynomial}\}$ there is an element in the set that dominates the sum.

So, the only remaining case to consider is that the last rule was an $\Sigma_i^B(\alpha)$-L^j-induction Rule, say

$$\frac{\Gamma(\vec{x}), \neg A(\vec{x}, a), A(\vec{x}, a+1)}{\Gamma(\vec{x}), \neg(s(\vec{x}, |\vec{\mathfrak{X}}|) = |t(\vec{x}, |\vec{X}|)|_j), \neg A(\vec{x}, 0), A(\vec{x}, s)}$$

with a an eigenvariable.

If $s(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}} \neq |t(\underline{\vec{g}}, \underline{\vec{f}})|_j^{\mathbb{N}}$ the conclusion contains T and can therefore be derived by an axiom. So we assume that $s(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}} = |t(\underline{\vec{g}}, \underline{\vec{f}})|_j^{\mathbb{N}}$. By Lemma 6.1.8 there is a polynomial $\mathfrak{r}$, such that $s(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}} \leq \log^{(j)}(\mathfrak{r}(\vec{f}, \vec{g}))$. As for any two polynomials there is another polynomial that dominates both, we can, without loss of generality assume that this is the same as the polynomial $\mathfrak{r}$ obtained by the induction hypothesis.

By induction hypothesis we have proofs

$$\vdash^{c\log^{(j+1)}(\mathfrak{r}(\vec{f},\vec{g}))+C}_{\Sigma_i^q(\alpha);\mathfrak{p}(\vec{f},\vec{g}),\mathfrak{q}(\vec{f},\vec{g})} \|\Gamma(\underline{\vec{g}})\|_{\vec{f}}, \neg\|A(\underline{\vec{g}}, \underline{k})\|_{\vec{f}}, \|A(\underline{\vec{g}}, \underline{k+1})\|_{\vec{f}}$$

for $0 \leq k \leq s(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}}$. By Lemma 6.3.1 we obtain therefore a derivation

$$\vdash^{c\log^{(j+1)}(\mathfrak{r}(\vec{f},\vec{g}))+C+\log(s(\underline{\vec{g}},\underline{\vec{f}})^{\mathbb{N}})}_{\Sigma_i^q(\alpha);\mathfrak{p}(\vec{f},\vec{g}),\mathfrak{q}(\vec{f},\vec{g})} \|\Gamma(\underline{\vec{g}})\|_{\vec{f}}, \neg\|A(\underline{\vec{g}}, \underline{0})\|_{\vec{f}}, \|A(\underline{\vec{g}}, s(\underline{\vec{g}}, |\vec{\mathfrak{X}}|))\|_{\vec{f}}$$

which, by weakening with F gives the desired conclusion. As for the height, we calculate that $\log(s(\underline{\vec{g}}, \underline{\vec{f}})^{\mathbb{N}}) \leq \log(\log^{(j)}(\mathfrak{r}(\vec{f}, \vec{g}))) = \log^{(j+1)}(\mathfrak{r}(\vec{f}, \vec{g}))$. Hence we can use the constants $c+1, C$ and the claim follows. □

Corollary 6.3.4. *For every $V^{i/j}(\alpha)$-proof of Γ with only Σ_{i+1}^B cuts, in particular for every free-cut free proof, there is a dynamic proof $\mathfrak{d}$ such that*

$$\mathfrak{d} \vdash^{\mathcal{O}(\log^{(j+1)})}_{\Sigma_i^q;\mathfrak{n}^{\mathcal{O}(1)},\mathfrak{n}^{\mathcal{O}(1)}} [\![\Gamma]\!]$$

Proof. Use Theorem 6.3.3. □

6.4 Translating Unrelativised Computation

Every method has its blind spots. In standard proof theory, for example, it is a folklore result that true Π_1^0-formulae can be added without affecting the height of the corresponding semi-formal proofs, and hence without affecting the proof theoretic ordinal.

As our measure of "sequentiality" heavily relies on sequential queries to the oracle α, it is no surprise that it cannot appreciate the strength of unrelativised computation. Formally, this is reflected in the observation, that such computations translate into constant-height polynomial-size $\mathsf{AC}^\star$-proofs. We now show this for unrelativised first-order formulae without free set-variables and for the naive computation of the transitive closure. Even though we will not make use of this fact, it should be obvious to the reader that these proofs trivially generalise to other unrelativised polynomial-time computation.

Proposition 6.4.1. *If $A(\vec{x})$ is a true, bounded, first-order $\mathcal{L}_2(\alpha)$-formula without free set-variables, then $\|A(\underline{\vec{n}})\|$ has constant-height AC^0-Tait proofs.*

Proof. Induction on A. If $A(\underline{\vec{n}})$ is a true equation or inequation, it will translate into T. If $A(\underline{\vec{n}}) \equiv \forall x < tB(x, \underline{\vec{n}})$ then, for all $m < t^{\mathbb{N}}$, the formula $B(\underline{m}, \underline{\vec{n}})$ is true. Hence we have constant height proofs for $\|B(\underline{m}, \underline{\vec{n}})\|$; a use of the $\bigwedge_{t^{\mathbb{N}}}$-rule yields a derivation of $\|\forall x < t.B(x, \underline{\vec{n}})\|$. The other connectives can be handled similarly. □

We'll know define the formula $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$ which expresses the property that, in the graph with adjacency matrix $(p_{i,j})_{i,j}$, there is a path of length at most ℓ from 0 to k.

Definition 6.4.2 ($\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$)**.** For $p_{00}, \ldots, p_{n-1,n-1}$ a list of n^2 propositional variables, $0 \leq k, \ell < n$ we define the propositional formula $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$ as

$$\begin{aligned}\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k) = \exists_{(\ell+1)n}\vec{q}\,. \quad & (q_{0,0} \leftrightarrow \mathrm{T}) \\ \wedge \quad & \textstyle\bigwedge_{j \neq 0}(q_{0,j} \leftrightarrow \mathrm{F}) \\ \wedge \quad & \textstyle\bigwedge_{i<\ell,j}(q_{i+1,j} \leftrightarrow (q_{i,j} \vee \bigvee_k (q_{i,k} \wedge p_{k,j}))) \\ \wedge \quad & q_{\ell,k}\end{aligned}$$

where all the conjunctions are to be read as one big conjunction.

Remark 6.4.3. Immediately by Definition 6.4.2 we note that $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$ is α-free. Hence the liberal comprehension rule of $\mathsf{AC}^\star$-Tait is available to these class of formulae.

Also, the size of the formulae $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$ is polynomial in the sense that there is a natural number C such that for all n and $0 \leq k, \ell < n$ we have $\mathsf{sz}(\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)) \leq C \cdot n^2 \cdot \ell + C'$.

Lemma 6.4.4. *There are constant height* $\mathsf{AC}^\star$*-Tait proofs of the following statements which are the usual defining equations of* $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},\ell,k)$*.*

$$\begin{aligned}
\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},0,0) &\leftrightarrow \mathrm{T}\\
\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},0,k+1) &\leftrightarrow \mathrm{F}\\
\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},\ell+1,k) &\leftrightarrow \Big(\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},\ell,k) \vee \textstyle\bigvee_j(\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p},\ell,j)\wedge p_{j,k})\Big)
\end{aligned}$$

Proof. The first two equivalences are simple, hence we concentrate on the last one and showing both implications separately.

Let $F_\ell(\vec{q})$ be short for $(q_{0,0} \leftrightarrow \mathrm{T}) \wedge \bigwedge_{j\neq 0}(q_{0,j} \leftrightarrow \mathrm{F}) \wedge \bigwedge_{i<\ell,j}(q_{i+1,j} \leftrightarrow (q_{i,j} \vee \bigvee_k(q_{i,k}\wedge p_{k,j})))$, implicitly fixing $\vec{p}$. In other words, $\delta^{\mathsf{NL}}_{\mathrm{step}}(k,\ell,\vec{p}) \equiv \exists_{(\ell+1)n}\vec{q}.(F_\ell(\vec{q})\wedge q_{\ell,k})$. Note that $F_{\ell+1}(\vec{q},\vec{q}') \equiv F_\ell(\vec{q}) \wedge \bigwedge_j(q'_{\ell+1,j} \leftrightarrow (q_{\ell,j} \vee \bigvee_k(q_{\ell,k}\wedge p_{k,j})))$. Moreover note that the formulae $F_\ell(\vec{q})$ are of constant depth.

Consider the following derivation.

$$\dfrac{\dots \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg F_\ell(\vec{q}), F_\ell(\vec{q})} \quad \dfrac{}{\neg q_{\ell,j}, q_{\ell,j}}\wedge}{\neg q_{\ell,j}, \neg F_\ell(\vec{q}), F_\ell(\vec{q})\wedge q_{\ell,j}}\exists}{\neg q_{\ell,j}, \neg F_\ell(\vec{q}), \delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})} \quad \dfrac{}{\neg p_{j,k}, p_{j,k}}\wedge}{\neg p_{j,k}, \neg q_{\ell,j}, \neg F_\ell(\vec{q}), \delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})\wedge p_{j,k}}\bigvee}{\neg p_{j,k}, \neg q_{\ell,j}, \neg F_\ell(\vec{q}), \bigvee_j(\delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})\wedge p_{j,k})}\vee,\vee}{\neg p_{j,k}\vee\neg q_{\ell,j}, \neg F_\ell(\vec{q}), \bigvee_j(\delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})\wedge p_{j,k})} \quad \dots}{\bigwedge_j(\neg p_{j,k}\vee\neg q_{\ell,j}), \neg F_\ell(\vec{q}), \bigvee_j(\delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})\wedge p_{j,k})}\bigwedge$$

Now we introduce the abbreviation

$$G_{\ell,k} \;\equiv\; \bigvee_j(\delta^{\mathsf{NL}}_{\mathrm{step}}(\ell,j,\vec{p})\wedge p_{j,k})$$

and continue the proof as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{(continued)}}{\bigwedge_j(\neg p_{j,k}\vee\neg q_{\ell,j}), \neg F_\ell(\vec{q}), G_{\ell,k}} \quad \dfrac{}{\neg q_{\ell,k}, q_{\ell,k}}\wedge}{\neg q_{\ell,k}\wedge\bigwedge_j(\neg p_{j,k}\vee\neg q_{\ell,k}), \neg F_\ell(\vec{q}), G_{\ell,k}, q_{\ell,k}} \quad \dfrac{}{\neg q_{\ell+1,k}, q_{\ell+1,k}}\wedge}{q_{\ell+1,k}\wedge\neg q_{\ell,k}\wedge\bigwedge_j(\neg p_{j,k}\vee\neg q_{\ell,k}), \neg F_\ell(\vec{q}), G_{\ell,k}, q_{\ell,k}, \neg q_{\ell+1,k}}\vee}{\neg(q_{\ell+1,k}\leftrightarrow(q_{\ell,k}\vee\bigvee_j(q_{\ell,k}\wedge p_{j,k}))), \neg F_\ell(\vec{q}), G_{\ell,k}, q_{\ell,k}, \neg q_{\ell+1,k}}$$

We continue again.

$$\dfrac{\dfrac{}{\neg F_\ell(\vec{q}), F_\ell(\vec{q})} \quad \dfrac{\text{(continued)}}{\ldots, \neg q_{\ell+1,k}, q_{\ell,k}}}{\dfrac{\neg F_\ell(\vec{q}), G_{\ell,k}, \neg(q_{\ell+1,k} \leftrightarrow (q_{\ell,k} \vee \bigvee_j (q_{\ell,j} \wedge p_{j,k}))), \neg q_{k,\ell+1}, F_\ell(\vec{q}) \wedge q_{\ell,k}}{\neg F_\ell(\vec{q}), G_{\ell,k}, \neg(q_{\ell+1,k} \leftrightarrow (q_{\ell,k} \vee \bigvee_j (q_{\ell,j} \wedge p_{j,k}))), \neg q_{k,\ell+1}, \delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k)}\,\exists}\,\wedge$$

By three $\bigvee$-rules we obtain

$$G_{\ell,k}, \neg F_\ell(\vec{q}) \vee \bigvee_k (\neg(q'_{\ell+1,k} \leftrightarrow (q_{\ell,k} \vee \bigvee_j (q_{\ell,j} \wedge p_{j,k})))) \vee \neg q_{k,\ell+1}, \delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k)$$

and finally by an $\forall$-rule

$$G_{\ell,k}, \neg\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k)$$

This shows one of the implications.

The other implication reduces to providing derivations

- $\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}), \neg p_{j,k}$ and
- $\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k)$

for then we can take the derivation

$$\dfrac{\ldots \quad \dfrac{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}), \neg p_{j,k}}{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}) \vee \neg p_{j,k}}\,\vee,\vee \quad \ldots}{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \bigwedge_j (\neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}) \vee \neg p_{j,k})}\,\bigwedge$$

and obtain

$$\dfrac{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k) \quad \dfrac{\text{(continued)}}{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \bigwedge_j (\neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}) \vee \neg p_{j,k})}}{\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \bigwedge_j (\neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}) \vee \neg p_{j,k}) \wedge \neg\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell, k)}\,\wedge$$

First we provide a derivation for $\delta^{\mathsf{NL}}_{\text{step}}(\vec{p}, \ell+1, k), \neg\delta^{\mathsf{NL}}_{\text{step}}(\ell, j, \vec{p}), \neg p_{j,k}$. Let

$$\Delta = \neg(q'_{\ell+1,1} \leftrightarrow q_{\ell,1} \vee \bigvee_k (q_{\ell,k} \wedge p_{k,1})), \ldots, \neg(q'_{\ell+1,n} \leftrightarrow q_{\ell,n} \vee \bigvee_k (q_{\ell,k} \wedge p_{k,n}))$$

and consider the following fragment of a derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots \quad (*) \quad \dots}{\Delta, F_\ell(\vec{q}) \wedge \bigwedge_j (q'_{\ell+1,j} \leftrightarrow q_{\ell,j} \vee \bigvee_k (q_{\ell,k} \wedge p_{k,j})) \wedge q'_{\ell+1,k}, \neg F_\ell(\vec{q}), \neg q_{\ell,j}, \neg p_{j,k}}\,\forall}{\Delta, \delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell+1, k), \neg F_\ell(\vec{q}), \neg q_{\ell,j}, \neg p_{j,k}}\,\exists}{\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell+1, k), \neg F_\ell(\vec{q}), \neg q_{\ell,j}, \neg p_{j,k}}\,(\mathrm{comp})}{\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell+1, k), \neg F_\ell(\vec{q}) \vee \neg q_{\ell,j}, \neg p_{j,k}}\,\vee,\vee}{\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell+1, k), \neg \delta^{\mathsf{NL}}_{\mathrm{step}}(\ell, j, \vec{p}), \neg p_{j,k}}\,\forall$$

where we used the $\mathsf{AC}^\star$-specific comprehension rule.

We will now provide the derivations needed at the place marked (*). Those coming from the individual conjunctions in $F_\ell(\vec{q})$ can be handled by using the constant-height tertium-non-datur proofs for them and matching them with the disjunction $\neg F_\ell(\vec{q})$. The conjuncts of the form $q'_{\ell+1,j} \leftrightarrow q_{\ell,j} \vee \bigvee_k (q_{\ell,k} \wedge p_{k,j})$ can be handled by tertium non datur again, this time leaving an element of Δ. So what remains is the case $q'_{\ell+1,k}$. Here we note that

$$\neg(q'_{\ell+1,k} \leftrightarrow q_{\ell,k} \vee \bigvee_{k'} (q_{\ell,k'} \wedge p_{k',k})), \neg q_{\ell,j}, \neg p_{j,k}, q'_{\ell+1,k}$$

is trivial tautology that can be shown by a constant height proof.

The case $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell+1, k), \neg \delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, k)$ is completely similar, except that at the end the trivial tautology

$$\neg(q'_{\ell+1,k} \leftrightarrow q_{\ell,k} \vee \bigvee_{k'} (q_{\ell,k'} \wedge p_{k',k})), \neg q_{\ell,k}, q_{\ell+1,k}$$

remains, which also has a constant height proof. □

For the following proof note that Corollary 3.4.7 which was used in the proof of Theorem 6.2.9 has its analogue Proposition 4.4.4 for $\mathsf{AC}^\star$-Tait.

Lemma 6.4.5. *For every $\mathsf{VNL}(\alpha)$-proof of Γ with only $\Sigma^B_1(\alpha)$-cuts, in particular for every free-cut free proof, there are polynomial-size (in $\vec{f}, \vec{g}$) constant-height $\mathsf{AC}^\star$-proofs of $\|\Gamma(\underline{\vec{f}})\|_{\underline{\vec{g}}}$, for $\vec{f}, \vec{g} \in \mathbb{N}$.*

Proof. We build on Theorem 6.2.9, hence we only have to provide constant height proofs of $\exists \mathfrak{Y} \delta_{\mathrm{conn}}(a, \mathfrak{X}, Y)$ where $\delta_{\mathrm{conn}}(a, \mathfrak{X}, \mathfrak{Y})$ is the formula.

$$\begin{aligned}&\mathfrak{Y}(0,0) \wedge \forall x < a (x \neq 0 \rightarrow \neg \mathfrak{Y}(0,x)) \wedge \\ &\forall z < a \forall x < a \, [\mathfrak{Y}(z+1,x) \leftrightarrow (\mathfrak{Y}(z,x) \vee \exists y < a (\mathfrak{Y}(z,y) \wedge \mathfrak{X}(y,x)))]\end{aligned}$$

The proof will end in

$$\dfrac{\dfrac{\dfrac{\dots}{\Delta, \|\delta_{\mathrm{conn}}(\underline{n}, \mathfrak{X}, \mathfrak{Y})\|_{\langle n;n\rangle,g}}}{\Delta, \|\exists\mathfrak{Y}\delta_{\mathrm{conn}}(\underline{n}, \mathfrak{X}, \mathfrak{Y})\|_g}\bigvee,\exists}{\|\exists\mathfrak{Y}\delta_{\mathrm{conn}}(\underline{n}, \mathfrak{X}, \mathfrak{Y})\|_g}\,(\mathrm{comp})$$

where

$$\Delta = \{p^{\mathfrak{Y}}_{\langle i;j\rangle} \leftrightarrow \delta^{\mathsf{NL}}_{\mathrm{step}}(p^{\vec{\mathfrak{X}}}_{\cdot}, i, j) \mid 0 \leq i, j < n\}\ .$$

Note that there is no constraint about $|\mathfrak{Y}|$, hence we can choose $\mathfrak{Y}$ of canonical length $\langle n; n\rangle$ and have the witness at hand for the $\bigvee$-introduction.

We note that $\|\delta_{\mathrm{conn}}(\underline{n}, \mathfrak{X}, \mathfrak{Y})\|_{\langle n;n\rangle,g}$ is (essentially) one big conjunction and each conjunct is a certain relation between the $p^{\mathfrak{Y}}_{\langle i;j\rangle}$. In context Δ, constant-height proofs can easily be constructed from the constant height proofs provided by Lemma 6.4.4. □

Corollary 6.4.6. *$\mathsf{VL}(\alpha)$-proofs can be translated as well into constant height polynomial-size $\mathsf{AC}^\star$-proofs.*

Proof. Immediately from Lemma 6.4.5 and the fact that $\mathsf{VL}(\alpha) \subseteq \mathsf{VNL}(\alpha)$. □

Remark 6.4.7. It should be noted that—even though we only presented the computation of the transitive closure—the above scheme generalises to arbitrary unrelativised polynomial-time computations. In fact, the defining axiom of $\mathsf{VNL}(\alpha)$ was formalising the computation of the transitive closure as a polynomial time algorithm, rather than using the fact that $\mathsf{NL} \subseteq \mathsf{AC}^1$.

Consider an unrelativised polynomial-size circuit. By existentially quantifying the bits representing the values of the nodes of the lowest ℓ-layers, we have an unrelativised formula, similar to $\delta^{\mathsf{NL}}_{\mathrm{step}}(\vec{p}, \ell, j)$, describing the value of the j'th node in the ℓ'th layer. The simple one-step relation between these nodes can be verified by a constant height proof as above.

As, however, no further arithmetical systems are discussed in this thesis that are axiomatised by adding unrelativised computation to $V^0(\alpha)$, we do not spell out a full proof of this observation, but leave it with this remark.

7 The Sequential Strength of Theories

We have seen (in Section 4) a tight connection between the height of AC^0-Tait proofs and parallel computation time, as measured by circuits. We have also seen (in Section 6) how proofs in Bounded Arithmetic relate to dynamic AC^0-Tait proofs. We now show how these connections can be used to construct a measure of the "sequential strength" for arbitrary theories in the language $\mathcal{L}_2(\alpha)$. This measure has a strong relation to computational complexity, a relation also witnessed by the fact, that for theories constructed explicitly from complexity classes we obtain the correct growth rate. This strong relation of a measure that is well defined for arbitrary two-sorted theories should therefore be useful for other reverse mathematical investigations with computational complexity in mind.

It should also be noted that the measure presented here shows the same picture as do the dynamical ordinals [7] in the range where the latter are defined, that is, in the range from $\mathfrak{n}^{\mathcal{O}(1)}$ to $2^{\mathfrak{n}}$. To see that the two measures coincide for all the theories investigated so far, one should keep in mind, that dynamical ordinals compare growth rate with the *value* of numbers whereas the sequential strength compares growth rate with the *size* of numbers. That explains why for a theory with dynamic ordinal $f(\log(\mathfrak{n}))$ the sequential strength is $f(\mathfrak{n})$.

So, the present work extends previous approaches in two ways. First, the range of applicable theories has been extended from growth rates $\mathfrak{n}^{\mathcal{O}(1)} \ldots 2^{\mathfrak{n}}$ to the full range of $\mathcal{O}(1) \ldots 2^{\mathfrak{n}}$. Secondly, and more importantly, it provides a clear meaning to the strength measure. So far, it was just an arbitrary growth rate associated to formal theories without further interpretation. Now it can be understood as the height of a circuit representing the computational power inherent in the theory under investigation.

7.1 The Arithmetic Formulation of the Sequential Iteration Principle

Following the lines mentioned in the introduction to this section, we aim at an arithmetical formula that translates into the n, ℓ-iteration principle introduced in Definition 4.2.4, or at least something closely related. As it turns out, we cannot directly obtain $\exists_{4n}\vec{p}\vec{p}'\vec{q}\vec{q}'.\ \Phi_{n,\ell}(\vec{p},\vec{p}',\vec{q},\vec{q}')$; keeping Definitions 4.2.1 and 4.2.2 in mind, we note that a certain shuffling of the $4n$ bits is done, in order to obtain the arguments for the parameter α_{2n}. However, in two-sorted arithmetic, we don't have enough string terms to perform such a shuffling right away. Nevertheless, as there are only polynomially many calls to the parameter and we have the row function $\cdot^{[\cdot]}$, we can do the usual trick

to look at the whole "computation transcript". In other words, our principle claims the existence of a string where the rows contain all the oracle calls ever made in the whole computation. Then we can refer to the i'th oracle call by using just the row function, and the bit shuffling is moved to an arithmetical formula stating that the individual bits in the various calls are composed in the correct way.

Notation 7.1.1 ("$\mathfrak{T}' = \mathfrak{T}+1$")**.** In accordance with Notation 4.2.3 we write "$\mathfrak{T}' = \mathfrak{T} + 1$" for the first order formula

$$\exists i{<}n.((\forall j{<}i.\mathfrak{T}'(j))\wedge\neg\mathfrak{T}'(i)\wedge(\forall j{<}i.\neg\mathfrak{T}(j))\wedge\mathfrak{T}(i)\wedge(\forall i{<}j{<}n.(\mathfrak{T}'(j)\leftrightarrow\mathfrak{T}(j))))$$

with free variables $\mathfrak{X}$, $\mathfrak{Y}$ and an implicit "size parameter" n, understood from the context.

Next, we would like to have similar notation for "$f^{\mathfrak{X}}(0) = \mathfrak{Y}$", which is analogous to Notation 4.2.2. However, we cannot just write $\alpha(\mathfrak{X}, \mathfrak{Y})$, as the latter is not in our language! The same applies to "$f(\mathfrak{X}) = \mathfrak{Y}$", which was introduced in Notation 4.2.1. We solve this problem by having an implicit parameter $\mathfrak{T}$ that serves as scratch pad for the string $\mathfrak{X}\mathfrak{Y}$ or the strings $i, \mathfrak{X}$, respectively.

Definition 7.1.2 ("$f^{\mathfrak{X}}(0) =_{(\mathfrak{T})} \mathfrak{Y}$")**.** If $\mathfrak{T}$ is a string term of the form $\mathfrak{T} \equiv \mathfrak{Z}^{[\vec{t}]}$, we write "$f^{\mathfrak{X}}(0) =_{(\mathfrak{T})} \mathfrak{Y}$" for the formula

$$\begin{array}{ll} & \alpha\mathfrak{T} \\ \wedge & (\forall j < n(\mathfrak{T}(i) \leftrightarrow \mathfrak{Y}(j))) \\ \wedge & (\forall j < n(\mathfrak{T}(n+i) \leftrightarrow \mathfrak{X}(j))) \\ \wedge & \mathfrak{T}(2\cdot n) \\ \wedge & (\forall j < |\mathfrak{Z}|(\neg\mathfrak{T}(2\cdot n+1+j)) \end{array}$$

with the length parameter n understood.

Remark 7.1.3. In Definition 7.1.2 we made the assumption that the string term $\mathfrak{T}$ witnessing the equality be of a particular form $\mathfrak{T} \equiv \mathfrak{Z}^{[\vec{t}]}$. It should be noted that *any* string term is for that form for some $\mathfrak{Z}$. So the only point in making this assumption was that we are able to "pull off" the underlying variable of the string term in the last clause. Recall Remark 5.1.3 for a rationale why we do not have the length function on arbitrary string terms.

Lemma 7.1.4. *The quantified propositional formulae*

$$\|\exists\mathfrak{Z} < \langle\underline{\ell}; \underline{2n}\rangle\ "f^{\mathfrak{X}}(0) =_{(\mathfrak{Z}^{[\underline{\ell}]})} \mathfrak{Y}\ "\|_{n,n}$$

and

$$\text{“}f^{p^{\mathfrak{X}}_{n-1}\cdots p^{\mathfrak{X}}_0}(0) = p^{\mathfrak{Y}}_{n-1}\dots p^{\mathfrak{Y}}_0\text{ ”}$$

are logically equivalent. Moreover, this equivalence is witnessed by a polynomial (in n) size AC^0-Tait proof of constant height.

Proof. Tedious, but elementary. For the implication from the second to the first formula, we can just use the (reshuffled) $p^{\mathfrak{X}}_{n-1}\dots p^{\mathfrak{X}}_0 p^{\mathfrak{Y}}_{n-1}\dots p^{\mathfrak{Y}}_0$ as witnesses for the existential statement.

For the other implication, note that all the bits occurring under the α are fixed by the statements in “$f^{\mathfrak{X}}(0) =_{(\mathfrak{Z}^{[\ell]})}\ \mathfrak{Y}$”, hence we have all the equivalences needed for the string extensionality rules. To construct derivations from this, we use proofs, similar to those constructed in Lemma 6.2.6. □

Definition 7.1.5 (“$f(\mathfrak{Y}) =_{(\ell,\mathfrak{Z})}\ \mathfrak{X}$”)**.** We write “$f(\mathfrak{Y}) =_{(\ell,\mathfrak{Z})}\ \mathfrak{X}$” for the formula

$$\begin{aligned}
&\forall i{<}n(\mathfrak{Y}(i) \leftrightarrow \alpha(\mathfrak{Z}^{[\ell+i]}))\\
\wedge\ &(\forall i{<}n\forall j{<}n(\mathfrak{Z}^{[\ell+i]}(j) \leftrightarrow \mathfrak{X}(j)))\\
\wedge\ &(\forall i{<}n\forall j{<}n(\mathfrak{Z}^{[\ell+i]}(n+j) \leftrightarrow \mathfrak{Z}^{[\ell+n+i]}(j)))\\
\wedge\ &(\forall i{<}n(\mathfrak{Z}^{[\ell+i]}(2\cdot n))\\
\wedge\ &(\forall i{<}n\forall j{<}|\mathfrak{Z}|(\neg\mathfrak{Z}^{[\ell+i]}(2\cdot n+1+j))\\
\wedge\ &(\forall j{<}n(\neg\mathfrak{Z}^{[\ell+n]}(j)))\\
\wedge\ &(\forall i{<}n\text{“}\mathfrak{Z}^{[\ell+n+i+1]} = \mathfrak{Z}^{[\ell+n+i]}+1\text{”})
\end{aligned}$$

with the length parameter n understood.

Remark 7.1.6. Definition 7.1.5 formalises $\forall i < n(\mathfrak{Y}(i) \leftrightarrow \alpha(i,\mathfrak{X}))$ solving the problem of not having $\alpha(i,\mathfrak{X})$ in the language by use of the auxiliary variable $\mathfrak{Z}$. More precisely, we think of $\mathfrak{Z}$ as a list of strings $\mathfrak{Z}^{[0]}$, $\mathfrak{Z}^{[1]}$, …, where the strings from index ℓ onwards are the scratch pad needed.

More precisely, it is stated that the strings $\mathfrak{Z}^{[\ell+i]}$, for $0 \le i < n$ have the shape

$$\dots 0001\ \underbrace{[i]}_{\log(n)}\ \underbrace{[\mathfrak{X}]}_{n}$$

so that $\alpha(\mathfrak{Z}^{[\ell+i]})$ translates to $\alpha_{n+\log(n)}(i, p^{\mathfrak{X}}_{n-1}\dots p^{\mathfrak{X}}_0)$ as desired. To have the binary coding of i, easily available, the last to closes state that (the last n bits of) $\mathfrak{Z}^{[\ell+n+i]}$ code the number i.

Lemma 7.1.7. *The quantified propositional formulae*

$$\|\exists\mathfrak{Z} < \langle \underline{\ell} + \underline{2n}; \underline{2n}\rangle\ \text{“}f(\mathfrak{Y}) =_{(\underline{\ell},\mathfrak{Z})}\ \mathfrak{X}\text{ ”}\|_{n,n}$$

and

$$\text{“}f(p^{\mathfrak{Y}}_{n-1}\dots p^{\mathfrak{Y}}_0) = p^{\mathfrak{X}}_{n-1}\dots p^{\mathfrak{X}}_0\text{ ”}$$

are logically equivalent. Moreover, this equivalence is witnessed by a polynomial (in n) size AC^0-Tait proof of constant height.

Proof. Tedious, but elementary. Compare the remarks in the proof of Lemma 7.1.4. Remark 7.1.6 explains how the helper variables are to be interpreted. Note that the "counting to n" is not a problem, as all the formulae involved there are quantifier free, so the multi-cut rules saves the day (recalling that resolution is complete). □

As mentioned, we are interested in growth rates in the range $\mathcal{O}(1)$ to $2^{\mathfrak{n}}$. To speak about these growth rate within a theory, we take number variables as input and produce a sequence of bits as output. Recall Definition 2.1.38 that shows how we identify strings with (exponentially large) numbers.

Definition 7.1.8 (String Function)**.** A *string function* $F_n(i)$ is a Δ_0^B-formula with two distinguished variables n, thought of as the input, and i.

A string function $F_n(\cdot)$ *denotes* the function

$$\begin{array}{rcl} \mathbb{N} & \to & \{0,1\}^* \\ n & \mapsto & (F_{\underline{n}}(\underline{n-1}))^{\mathbb{N}} \dots (F_{\underline{n}}(\underline{0}))^{\mathbb{N}} \in \{0,1\}^n \ . \end{array}$$

Definition 7.1.9 (Arithmetical Iteration Formula)**.** If $F_n(\cdot)$ is a string function, the *arithmetical iteration formula* $\Phi_{F.(\cdot)}$ is the formula

$$\begin{array}{l} \Phi_{F_n(\cdot)}(\mathfrak{X},\mathfrak{X}',\mathfrak{Y},\mathfrak{Y}') \equiv \\ \exists \mathfrak{Z} {<} \langle \underline{2} + \underline{2} \cdot n; n \rangle [\\ \quad ((\forall i < n . \mathfrak{Y}(i) \leftrightarrow F_n(i)) \wedge \text{“} f^{\mathfrak{Y}}(0) =_{(\mathfrak{Z}^{[0]})} \mathfrak{X} \text{”}) \\ \vee \quad \neg \text{“} f^0(0) =_{(\mathfrak{Z}^{[0]})} 0 \text{”} \\ \vee \quad (\text{“} \mathfrak{Y}' = \mathfrak{Y} + 1 \text{”} \wedge \text{“} f^{\mathfrak{Y}}(0) =_{(\mathfrak{Z}^{[0]})} \mathfrak{X} \text{”} \wedge \\ \qquad \text{“} f(\mathfrak{X}') =_{(\underline{2},3)} \mathfrak{X} \text{”} \wedge \neg \text{“} f^{\mathfrak{Y}'}(0) =_{(\mathfrak{Z}^{[1]})} \mathfrak{X}' \text{”}) \ ] \end{array}$$

The *arithmetical iteration principle for $F.(\cdot)$* is the formula

$$\exists \mathfrak{X}\mathfrak{X}'\mathfrak{Y}\mathfrak{Y}' < n{+}1 \, \Phi_{F_n(\cdot)}(\mathfrak{X},\mathfrak{X}',\mathfrak{Y},\mathfrak{Y}')$$

Lemma 7.1.10. *The quantified propositional formulae*

$$\|\exists \mathfrak{X} {<} \underline{n{+}1} \exists \mathfrak{X}' {<} \underline{n{+}1} \exists \mathfrak{Y} {<} \underline{n{+}1} \exists \mathfrak{Y}' {<} \underline{n{+}1} \Phi_{F_{\underline{n}}(\cdot)}(\mathfrak{X},\mathfrak{X}',\mathfrak{Y},\mathfrak{Y}')\|$$

and

$$\exists_{4n} \vec{p}\vec{p}'\vec{q}\vec{q}' . \ \Phi_{n,(F_n(\cdot))^{\mathbb{N}}}(\vec{p},\vec{p}',\vec{q},\vec{q}') \ .$$

are logically equivalent and the equivalence is witnessed by a polynomial (in n) size AC^0-Tait proof of constant height.

Here $\Phi_{n,\ell}$ is the iteration formula introduced in Definition 4.2.4 and $(F_n(\cdot))^{\mathbb{N}}$ is the function denoted by $F.(\cdot)$ at the value n.

Proof. Tedious, but elementary. Builds on the proofs constructed in Lemmata 7.1.4 and 7.1.7. □

Definition 7.1.11 (The Sequential Strength $\mathsf{seq}(T)$ of a Theory T)**.** If T is a theory in $\mathcal{L}_2(\alpha)$, we define

$$\begin{aligned}\mathsf{seq}(T) = \{\mathfrak{f}\colon \mathbb{N} \to \mathbb{N} \mid\ & \text{there is some string function } F_n(\cdot) \\ & \text{such that } \mathfrak{f} \leq^{\mathrm{e}} (F_{\mathfrak{n}}(\cdot))^{\mathbb{N}} \text{ and} \\ & T \vdash \exists \mathfrak{X}\mathfrak{X}'\mathfrak{Y}\mathfrak{Y}' < n{+}1\, \Phi_{F_n(\cdot)}(\mathfrak{X}, \mathfrak{X}', \mathfrak{Y}, \mathfrak{Y}') \ \}\end{aligned}$$

that is the set of all functions that are eventually dominated by some string function for which T proves the arithmetical iteration principle.

7.2 The Sequential Strength of $V^0(\alpha)$

The theory $V^0(\alpha)$ is the basis for all the theories under consideration, that is, all these theories are extensions of $V^0(\alpha)$. In particular, as they prove more theorems, their sequential strength exceeds that of $V^0(\alpha)$. So the sequential strength of $V^0(\alpha)$ is the natural lower bound down to which our method yields useful results. Fortunately it is $\mathcal{O}(1)$.

Theorem 7.2.1. $\mathsf{seq}(V^0(\alpha)) = \mathcal{O}(1)$

Proof. Arguing informally in $V^0(\alpha)$ it is easy to show that the function given by an oracle can be iterated a constant number of times. This shows the lower bound.

As for the upper bound of the sequential strength, assume a $V^0(\alpha)$ proof of the sequential iteration principle. By Corollary 6.2.10 this translates to a constant height AC^0-Tait proof of a formula which, by Lemma 7.1.10, is equivalent to the with the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle introduced in Definition 4.2.4. Here $\mathfrak{f}$ is the function denoted by $F_{\cdot}(\cdot)$.

Since the equivalence is witnessed by a constant height proof and since the formulae involved are Σ^q_1-formulae we can appeal to Corollary 3.4.7 and obtain a constant height proof of the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle, which has still polynomial width. Therefore, Corollary 4.2.18 shows that $\mathfrak{f}$ is bounded by a constant. This finishes the proof. □

7.3 The Sequential Strength of $\mathsf{VNL}(\alpha)$

Theorem 7.3.1. $\mathsf{seq}(\mathsf{VNL}(\alpha)) = \mathcal{O}(1)$

Proof. Since $\mathsf{VNL}(\alpha)$ extends $V^0(\alpha)$ it is easy to show, that the function given by the oracle can be iterated a constant number of times, hence the lower bound.

For the upper bound a assume a $\mathsf{VNL}(\alpha)$ proof of the sequential iteration principle. By Lemma 6.4.5 this translates into a family of constant height $\mathsf{AC}^\star$-proofs of a formula which, by Lemma 7.1.10, is equivalent to the with the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle introduced in Definition 4.2.4. Here $\mathfrak{f}$ is the function denoted by $F.(\cdot)$.

Since the equivalence is witnessed by a constant height proof and since the formulae involved are Σ^q_1-formulae we can appeal to Corollary 4.4.4 and obtain a constant height proof of the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle, which has still polynomial width. Therefore, Corollary 4.4.6 shows that $\mathfrak{f}$ is bounded by a constant. This finishes the proof. □

Corollary 7.3.2. $\mathsf{seq}(\mathsf{VL}(\alpha)) = \mathcal{O}(1)$

Proof. Immediate from Theorems 7.2.1 and 7.3.1, since $V^0(\alpha) \subseteq \mathsf{VL}(\alpha) \subseteq \mathsf{VNL}(\alpha)$. □

It seems a bit disappointing that our new measure fails to separate the theories for $\mathsf{AC}^0(\alpha)$ and NL^α. Note, however, that these complexity classes have complete problems that are not relativised. In other words, the axiomatisations of our theories between $V^0(\alpha)$ and $\mathsf{VNL}(\alpha)$ only differ in the unrelativised parts. In particular, any separation would also separate the corresponding unrelativised theories. Moreover, as the Σ^B_1-definable functions are precisely the corresponding unrelativised complexity classes [14, 20], a separation at the Σ^B_1-level would even unconditionally separate the corresponding complexity classes. It is worth noting, that our arithmetical strength measure introduced in Definition 7.1.11 is entirely based on the $\Sigma^B_1(\alpha)$-consequences of the theory in question.

A separation of AC^0 from L is known, since AC^0 cannot compute parity [24]. However, it is still an open problem whether $\mathsf{AC}^0(6)$ is separated from polynomial time. This open problem is, unfortunately, not solved in this thesis either.

Nevertheless, the author believes that Theorem 7.3.1 still has something to say. Recall the tight correspondence between sequential iteration and parallel time, as measured by circuit height (Lemma 4.3.3 and Theorem 4.3.9). Keeping this correspondence in mind, one can rephrase Theorem 7.3.1 as "Whatever the difference between AC^0 and NL is, it has nothing to do with the ability of doing more computations in sequence".

7.4 The Sequential Strength of $V^{i/j}(\alpha)$

Theorem 7.4.1. $\mathsf{seq}(V^{i/j}(\alpha)) \subset 2_i(\mathcal{O}(\log^{j+1}))$

Proof. Assume a $V^{i/j}(\alpha)$ proof of the sequential iteration principle. By Corollary 6.3.4 this translates to an AC^0-Tait proof with $\Sigma^q_i(\alpha)$-cuts of height $\mathcal{O}(\log^{j+1})$, again of a formula which, by Lemma 7.1.10, is equivalent to the with the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle introduced in Definition 4.2.4. Here $\mathfrak{f}$ is the function denoted by $F_{\cdot}(\cdot)$.

Using cut-elimination provided by Proposition 3.4.10, we get a cut-free AC^0-Tait proof of height $2_i(\mathcal{O}(\log^{j+1}))$.

Since the equivalence of the proved formula and the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle is witnessed by a constant height proof and since the formulae involved are Σ^q_1-formulae we can appeal to Corollary 3.4.7 and obtain proof of height $2_i(\mathcal{O}(\log^{j+1}))$ of the $\mathfrak{n}, \mathfrak{f}(\mathfrak{n})$-iteration principle, which has still polynomial width. Therefore, Corollary 4.2.18 shows that $\mathfrak{f} \in 2_i(\mathcal{O}(\log^{j+1}))$. This finishes the proof. □

It should be noted that, even though true for general i and j, Theorem 7.4.1 yields non-trivial results only for $j+2 > i$, for otherwise the trivial $2^{\mathfrak{n}}$ bound is more strict.

We will now show that the bound in Theorem 7.4.1 in tight in the cases where $2_i(\mathcal{O}(\log^{j+1}))$ is eventually dominated by $2^{\mathfrak{n}}$. To avoid technical complication we presuppose an underlying size parameter n (in the form of a free variable) and we tacitly assume all occurring numbers to be bound by 2^n. Of course, such large numbers are represented as strings.

Moreover, we write unbounded string quantifiers as a shorthand for string quantifiers with length bounded by n. To avoid ambiguities we hereby adopt the convention that the rest of this subsection does not use any unbounded string quantifiers. As a matter of fact, it does not use any unbounded number quantifiers either.

Exponentiation $2^{\mathfrak{X}}$ can be defined as a string function by stating that the i'th bit of $2^{\mathfrak{X}}$ is set if and only if $\mathfrak{X}$ represents i. The latter is the case, if, for all j, the j'th bit of i is set in the sense of $\mathrm{BIT}(i,j)$, see Definition 5.4.28, if and only if $\mathfrak{X}(j)$. As is well-known [24], multiplication $\mathfrak{X} \cdot \mathfrak{Y}$ in not in AC^0 and hence not first-order definable (in a bounded setting). Fortunately, being a simple polynomial-time computation, it is definable as a Δ^B_1-formula over any theory providing Σ^B_1-induction up to $\mathfrak{X}$. In fact, the pair of formulae can be chosen independent of the theory. As we will use multiplication only in situations where the first factor is small enough that it could be a number we define multiplication by the Δ^B_1-formula formalising repeated addition. In this way, the defining equations of multiplication easily follow

from the definition, even over $V^0(\alpha)$. As all formulae using multiplication will be of logical complexity Σ_1^B or higher, substituting in multiplication as a Δ_1^B-formula doesn't increase the logical complexity, as has been shown in Lemma 5.1.32. For $\mathfrak{X}$ a string variable and $C > 0$ a natural number (on the meta-level) we use $\mathfrak{X}^C$ as a shorthand for $\mathfrak{X}\cdot(\mathfrak{X}\cdot(\ldots\cdot(\mathfrak{X}\cdot\mathfrak{X})))$, with C factors, parenthesised to the right.

When arguing informally in theories we are quite lax about arithmetical relations (like associativity of string addition and the laws of exponentiation to base two). This is justified as all our theories extend V^0. A reader feeling uncomfortable with lax treatment should note that all the needed arithmetical properties do not refer to the oracle α. Therefore, they can safely be added as additional axioms; as seen in Section 6.4, unrelativised computations don't affect the upper bound on sequential iteration.

Definition 7.4.2 (Inductive Formula)**.** Let $\varphi(\mathfrak{X})$ be a formula with a distinguished string variable $\mathfrak{X}$ and possibly other free variables. We call $\varphi(\mathfrak{X})$ *inductive* if $\varphi(0)$ and $\forall\mathfrak{X}(\varphi(\mathfrak{X}) \to \varphi(\mathfrak{X}+1))$ hold; $\varphi(\mathfrak{X})$ is inductive in some theory, if that theory proves these properties.

Following an idea by Gentzen [27] we show how induction on more complicated formulae can be used to induct a longer distance.

Definition 7.4.3 (Jump $\varphi'(\mathfrak{X})$ of a formula)**.** Let $\varphi(\mathfrak{X})$ be a formula with a distinguished free variable $\mathfrak{X}$ and possibly other free variables. We define the *jump* φ' of φ to be the formula

$$\varphi'(\mathfrak{Y}) = \forall\mathfrak{X}[\varphi(\mathfrak{X}) \to \varphi(\mathfrak{X}+2^{\mathfrak{Y}})]$$

Remark 7.4.4. Immediately from Definition 7.4.3 we note that φ' is $\Pi_{i+1}^B(\alpha)$ if φ is $\Pi_i^B(\alpha)$ or $\Sigma_i^B(\alpha)$.

Lemma 7.4.5. *As a theorem of V^0, if φ is inductive, then so is φ'.*

Proof. Since φ is inductive, we have that $\forall\mathfrak{X}(\varphi(\mathfrak{X}) \to \varphi(\mathfrak{X}+1))$ which, by arithmetic, amounts to $\forall\mathfrak{X}(\varphi(\mathfrak{X}) \to \varphi(\mathfrak{X}+2^0))$. Hence $\varphi'(0)$.

We now show that $\varphi'(\mathfrak{Y})$ implies $\varphi'(\mathfrak{Y}+1)$. So let $\mathfrak{Y}$ be arbitrary and assume $\varphi'(\mathfrak{Y})$. We have to show $\varphi'(\mathfrak{Y}+1)$. Hence let $\mathfrak{X}$ be arbitrary and assume $\varphi(\mathfrak{X})$ in order to show $\varphi(\mathfrak{X}+2^{\mathfrak{Y}+1})$. From $\varphi'(\mathfrak{Y})$ and $\varphi(\mathfrak{X})$ we get $\varphi(\mathfrak{X}+2^{\mathfrak{Y}})$. Using $\varphi'(\mathfrak{Y})$ again, this time instantiation the outermost quantifier by $\mathfrak{X}+2^{\mathfrak{Y}}$, and using the just obtained $\varphi(\mathfrak{X}+2^{\mathfrak{Y}})$ we get $\varphi((\mathfrak{X}+2^{\mathfrak{Y}})+2^{\mathfrak{Y}})$ which, by arithmetic, shows the claim. □

Proposition 7.4.6. *Let φ be inductive. Then, as a theorem of V^0, the statement $\varphi'(n)$ implies $\varphi(2^n)$.*

Proof. Instantiating the outermost quantifier in $\varphi'(n)$ by 0 we obtain $\varphi(0) \rightarrow \varphi(2^n)$. Since φ is inductive we know $\varphi(0)$. Hence the claim. □

Inspecting Theorem 7.4.1 we note that the important difference between theories like $V^{i/j}(\alpha)$ and $V^{i+1/j+1}(\alpha)$ is the place where the constant (of the $\mathcal{O}$-notation) appears. By looking at the proof of Theorem 6.3.3 we note that the constant essentially arises as the number of nested inductions.

So, in order to exactly match the upper bound, we have to construct a proof by induction on the constant. This will be provided by the meta-induction in the next lemma.

Lemma 7.4.7. *Consider a theory that proves $\Pi^B_i(\alpha)$-induction up to N. Let $\varphi(\mathfrak{X})$ be $\Pi^B_i(\alpha)$, and $C > 0$ a natural number. Then the following is a theorem of that theory.*

Assume that $\varphi(\mathfrak{X})$ is inductive. Then $\varphi(N^C)$.

Proof. We argue by (meta) induction on C. If $C = 1$ the claim follows immediately from the induction principle. So let $C > 1$ and argue informally in the theory.

Let $\psi(\mathfrak{X})$ be the formula

$$\psi(\mathfrak{X}) \equiv \varphi(N \cdot \mathfrak{X})$$

for which we show that it is inductive. Note that is has the same logical complexity as φ. Hence the meta-induction hypothesis provides us with $\psi(N^C)$, that is $\varphi(N^{C+1})$. Hence the claim.

Since φ is inductive, we have $\varphi(0)$ which is the same as $\psi(0)$.

Finally we have to show that $\psi(\mathfrak{X})$ implies $\psi(\mathfrak{X}+1)$. So assume $\psi(\mathfrak{X})$. We have to show that $\psi(\mathfrak{X}+1)$ holds. To do so, we show by induction on i that $\varphi(\mathfrak{X} \cdot N + i)$ holds. Since our induction goes till N we then get $\varphi(\mathfrak{X} \cdot N + N)$, hence the claim. $\varphi(\mathfrak{X} \cdot N + 0)$ holds by our assumption $\psi(\mathfrak{X})$. Moreover, since φ is inductive $\varphi(\mathfrak{X} \cdot N + i)$ implies $\varphi(\mathfrak{X} \cdot N + i + 1)$. This finishes the proof. □

Theorem 7.4.8. *Let φ be inductive and $\Sigma^B_i(\alpha)$. Let $k \geq 0$ and $C > 0$ be a natural numbers. Then $\Sigma^B_{i+k}(\alpha)$-induction up to N implies*

$$\varphi(2_k(N^C))$$

Note that this in particular implies that for φ a $\Sigma^B_1(\alpha)$-formula and $i > 0$ it is the case that $V^{i/j}(\alpha)$ proves $\varphi(2_{i-1}([\log^j(n)]^C))$ where n is a free number variable that, as usual, serves as a size parameter. Since $2_{i-1}([\log^j(n)]^C)$ has the same growth rate as

$$2_i(C \cdot \log^{j+1}(n))$$

this matches the upper bound (for iteration as a $\Sigma^B_1(\alpha)$-principle).

Proof. We first show how the last claim can be obtained from the previous ones. Recall that $V^{i/j}(\alpha)$ provides $\Pi^B_i(\alpha)$ induction up to $\log^j(n)$. So the first claim asserts that $V^{i/j}(\alpha)$ proves $\varphi(2_{i-1}((\log^j n)^C))$. Using that $2^{\log}$ and the identity have the same growth rate, the argument has the growth rate $2_i(\log([\log^j(n)]^C)) = 2_i(C \cdot \log^{j+1}(n))$.

So, let's show the main claim. If φ is $\Sigma^B_i(\alpha)$ then iterated use of Remark 7.4.4 shows that the k-fold jump $\varphi^{(k)}$ is $\Pi^B_{i+k}(\alpha)$ for $k \geq 1$ and $\Sigma^B_i(\alpha)$ for $k = 0$. In any case we can do induction on $\varphi^{(k)}$, using Proposition 5.5.8.

So by Lemma 7.4.7 we obtain $\varphi^{(k)}(N^C)$ and therefore, by Lemma 7.4.6, we get $\varphi(2_k(N^C))$ as desired. □

Corollary 7.4.9. *Let $i, j \in \mathbb{N}$ be natural numbers and $j + 2 > i$. Then*

$$\mathsf{seq}(V^{i/j}(\alpha)) = 2_i(\mathcal{O}(\log^{j+1}))$$

Proof. The upper bound is provided by Theorem 7.4.1 The lower bound is provided by Theorem 7.4.8, noting that the arithmetical iteration formula is $\Sigma^B_1(\alpha)$ and, over V^0, inductive. □

The separation of theories obtained in this way holds for arbitrary large j, giving a strict hierarchy of theories with less and less induction. This is possible as $V^0(\alpha)$ proofs translate to constant height proofs; there is no "ground noise" due to the method. This is different from dynamical ordinal analysis [7], where the need for complete cut-elimination gives rise to a polynomial (i.e., poly-logarithmic in the value) height increase due to first-order (i.e., sharply bounded) cut-elimination.

7.5 Back to the Beginning: Circuits Again

There is one class of arithmetical theories in the literature not yet discussed in this thesis. The reason is that we, implicitly, used these theories as calibration standard for our strength measure. These are theories that axiomatise, over the base theory $V^0(\alpha)$, that circuits of a certain shape can be evaluated. More precisely, the axiomatisation is, over $V^0(\alpha)$ equivalent to saying that for all circuits obeying a certain shape restriction (in the underlying size parameter in the form of a free number variable), the circuit-evaluation formula (Definition 4.3.1) holds true.

Most prominently, the theories $\mathsf{VAC}^k(\alpha)$ axiomatise [4, 46] that this is the case for relativised $\mathsf{AC}^k(\alpha)$-circuits, as defined in Definition 2.4.5. Circuits with unbounded fan-in are, as shown in Section 4.3, in one-to-one correspondence with proof heights in propositional logic. Therefore (using our results about $V^0(\alpha)$ as given) we immediately that $\mathsf{seq}(\mathsf{VAC}^k(\alpha)) = \mathcal{O}(\log^k)$, confirming again our intuitive understanding of the sequential strength.

8 Conclusions and Future Work

In this thesis, we have studied various aspects computation relative to an oracle. A main focus was con complexity classes within Polynomial Time.

First, we have introduced relativised versions of the complexity classes L, NL, AC^k, and NC^k in such a way that all the known inclusions and closure properties are preserved. Besides closure under composition, this includes relativised versions of Immerman-Szelepcseényi's and Savitch's theorem. Such definitions did not exist in the literature before.

We have identified iteration as a suitable principle to characterise a notion of "parallel time" for relativised computation. For Boolean circuits, the amount of iteration that can be carried out by a circuit is precisely the height of that circuit—and for circuits, height is the agreed notion of time. The principle, however is a computational task that can be posed to any model of computation that has access to an oracle. So we have obtained a means to, in a sensible way, speak of "time" for a relativised computational complexity class—no matter how it is defined. It need not even be defined by a machine model.

We have then introduced formal theories in the style of Bounded Arithmetic, that correspond to the relativised complexity classes studied. The correspondence is a very tight one, in several ways. On the one hand, the defining axioms (over our weak base theory $V^0(\alpha)$) are precisely the statements that the complete problems for the complexity classes in question have a solution. This has the consequence that the provably recursive functions of that theory are precisely those that belong to the corresponding relativised complexity class. These results have been first presented in a joint article [4] with Stephen Cook and Phuong Nguyen. We have also included the theories $V^{i/j}(\alpha)$ that recast in the two-sorted settings the theories of Bounded Arithmetic that are based on restricted induction.

Based on the above results, we have devised a taxonomy for weak formal theories, like the ones we have developed for the complexity classes studied. The measure was again the iteration principle. Note that "iterating a function" is a concept abstract enough so that we can formulate it first-order logic, provided a bit of coding is available. Basing our investigations on the iteration principles allowed us to use all our previous results; in particular, our measure is well calibrated. For theories based on the Boolean-circuit complexity classes we obtain their height as "sequential strength". For example, the sequential strength of $\mathsf{VAC}^k(\alpha)$ is the set of functions $\mathcal{O}(\log^k)$. In particular, the theories for relativised Boolean circuits are completely separated. So we obtain a complete picture down to the level of $\mathsf{AC}^0(\alpha)$. The only related measure that existed before, Arnold Beckmann's "dynamic or-

dinal" [7], could not look within the complexity class polynomial time. This is due to the need of a complete cut-elimination which creates a polynomial overhead—and thus dominates, and therefore hides, all smaller effects. But the sequential strengths is not merely a new, more expressive measure. It also properly extends the dynamic ordinal. Wherever the dynamic ordinal gives "proper" results, i.e., results that are not an artefact to the definition—or in other words, for all theories where matching lower bounds exists—both measures yield the same growth rate. This extra awareness, and the fact that the growth rate obtained for theories as a clear computational meaning in terms of Boolean circuits, makes it potentially useful for classifying mathematical arguments according to their computational strengths. In a recent program known as "low-level reverse mathematics" [46] for mathematical statements, like a discrete version of the Jordan curve theorem [47], the minimal theory is identified that is able to prove this principle.

On the way to the results on iteration, a lot of propositional logic had to be developed. All the bound presented are obtained from bounds in propositional logic after a suitable translation and appropriate cut-elimination. For this approach to work out, it was necessary to develop a calculus AC^0-Tait for propositional logic relative to an oracle, a concept new to the literature. Moreover, the calculus is designed in such a way, that the height of derivations in this calculus has an intimate connection to Boolean circuits. As such a calculus seemed of independent interest it has first been presented in a joint article [2] with Arnold Beckmann.

Even though this thesis gives satisfying answers to many questions, some directions are still open for future research. The most prominent is that of the need of relativisation. All our results refer to computation relative to an oracle. Moreover, since unrelativised computation can be proved to exist with constant height derivations, it seems that relativisation is essential for our method. On the other hand, that situation is similar to the situation in traditional proof-theory after an ordinal-analysis: for a theory, the provable total well-orders have been determined, and being a well-order is an inherently second-order concept. Nevertheless, such a proof-theoretic analysis was often only the first and key step towards meaningful unprovable statements in the language of arithmetic. Goodstein [28] obtained such a statement by coding appropriate fundamental sequences for the hard ordinal notation system arithmetically. This obviously is a quite general method. More indirect methods [49] identify hard combinatorial principles like finite versions of the Ramsey Theorem. The latter is also an active area of investigation [36] in Bounded Arithmetic—however, again in a relativised setting. Even though Ramsey's principle, speaking of a graph and a partition of that graph, can be formalised in Bounded Arithmetic, using the oracle to describe the partition

allows to speak about bigger graphs and partitions thereof. Unfortunately, this extra size is needed to get unprovability results at all.

A different approach that is currently being investigated in the context of propositional proof complexity [37, 38] is that of proof complexity generators. Recall the way we used the oracle in our iteration principle. It essentially served as a hard-to-predict function. The idea now is to investigate if a more explicitly given function, like a pseudo random number generator, can be still hard enough to predict—at least under some plausible assumptions. But since the setting in which proof complexity generators are usually studied is slightly different, a detailed comparison is still to be carried out.

Finally, it should be noted, there is some benefit in applying the proof-theoretic method directly to unrelativised theories, i.e., useful results can be obtained by propositional translation and cut-elimination of theories not mentioning an uninterpreted predicate. In a recent joint article [3] it was used to reobtain in a *uniform* way the known characterisations [10, 11, 35] of the total relations of S^i_2 definable by Σ^b_{i-1}-formulae, Σ^b_i-formulae, and Σ^b_{i+1}-formulae.

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[2] K. Aehlig and A. Beckmann. Propositional logic for circuit classes. In Duparc and Henzinger [22], pages 512–526.

[3] K. Aehlig and A. Beckmann. On the computational complexity of cut-reduction. In *Proceedings of the Twentythird Annual IEEE Symposium on Logic in Computer Science (LICS '08)*, pages 284–293, June 2008.

[4] K. Aehlig, S. A. Cook, and P. Nguyen. Relativizing small complexity classes and their theories. In Duparc and Henzinger [22], pages 374–388.

[5] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, Dec. 1975.

[6] A. Beckmann. *Separating fragments of bounded arithmetic*. PhD thesis, Westfälische Wilhelms-Universität Münster, 1996.

[7] A. Beckmann. Dynamic ordinal analysis. *Archive for Mathematical Logic*, 42:303–334, 2003.

[8] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[9] W. Buchholz. An approach to variable bindung using de Bruijn indices and named variables. Manuscript, available at `http://www.mathematik.uni-muenchen.de/~buchholz/`, Aug. 2002.

[10] S. R. Buss. *Bounded Arithmetic*. Bibliopolis, 1986.

[11] S. R. Buss and J. Krajíček. An application of boolean complexity to separation problems in bounded arithmetic. *Proceedings of the London Mathematical Society*, 69(3):1–27, 1994.

[12] S. A. Cook. Towards a complexity theory of synchronous parallel computation. *L'Enseignement Mathématique*, 27:99–124, 1981.

[13] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[14] S. A. Cook. Theories for complexity classes and their propositional translations. In J. Krajíček, editor, *Complexity of computations and proofs*, Quaderni die Matematica, pages 175–227. Dipartimento di Matematica, Seconda Universitá degli Studi di Napoli, 2003.

[15] S. A. Cook and A. Kolokolova. A second-order system for polytime reasoning based on Grädel's theorem. *Annals of Pure and Applied Logic*, 124:193–231, 2003.

[16] S. A. Cook and A. Kolokolova. A second-order theory for NL. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science (LICS '04)*, pages 398–407, July 2004.

[17] S. A. Cook and T. Morioka. Quantified propositional calculus and a second-order theory for NC^1. *Archive for Mathematical Logic*, 44(6):711–749, 2005.

[18] S. A. Cook and P. Nguyen. Foundations of proof complexity: Bounded arithmetic and propositional translations. draft of a book, available at `http://www.cs.toronto.edu/~sacook/csc2429h/book/`.

[19] S. A. Cook and P. Nguyen. VTC^0: A second-order theory for TC^0. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science (LICS '04)*, pages 378–387, July 2004.

[20] S. A. Cook and P. Nguyen. Theories for TC^0 and other small complexity classes. *Logical Methods in Computer Science*, 2(1), 2006.

[21] S. A. Cook and R. A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1), Mar. 1979.

[22] J. Duparc and T. Henzinger, editors. *Proceedings of the sixteenth Annual Conference on Computer Science and Logic*, volume 4646 of *Lecture Notes in Computer Science*. Springer Verlag, Sept. 2007.

[23] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, editor, *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*. American Mathematical Socienty, 1974.

[24] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[25] G. Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39:176–210, 1935.

[26] G. Gentzen. Untersuchungen über das logische Schließen. II. *Mathematische Zeitschrift*, 39:405–431, 1935.

[27] G. Gentzen. Beweisbarkeit und Unbeweisbarkeit von Anfangsfällen der transfiniten Induktion. *Mathematische Annalen*, 119:149–161, 1943.

[28] R. L. Goodstein. On the restricted ordinal theorem. *The Journal of Symbolic Logic*, 9(2):31–41, 1944.

[29] Y. Gurevich and H. R. Lewis. A logic for constant-depth circuits. *Information and Control*, 61:65–74, 1984.

[30] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[31] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778, Aug. 1987.

[32] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[33] N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer Verlag, 1999.

[34] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.

[35] J. Krajíček. Fragments of bounded arithmetic and bounded query classes. *Transactions of the American Mathematical Society*, 338(2):587–598, 1993.

[36] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

[37] J. Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1–3):123–140, 2001.

[38] J. Krajíček. Tautologies from pseudo-random generators. *The Bulletin of Symbolic Logic*, 7(2):197–212, 2001.

[39] J. Krajíček and P. Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 36:29–46, 1990.

[40] J. Krajíček, P. Pudlák, and G. Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52(1–2):143–153, 1991.

[41] R. E. Ladner and N. A. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.

[42] L. Libkin. *Elements of Finite Model Theory*. Springer Verlag, 2004.

[43] S. Lindell. A purely logical characterization of circuit uniformity. In *Structure in Complexity Theory Conference*, pages 185–192, June 1992.

[44] J. C. Martin. *Introduction to languages and the theory of computation*. McGraw-Hill, Inc., 1991.

[45] T. Morioka. *Logical Approaches to the Complexity of Search Problems: Proof Complexity, Quantified Propositional Calculus, and Bounded Arithmetic*. PhD thesis, Department of Computer Science, University of Toronto, 2005.

[46] P. Nguyen. *Bounded Reverse Mathematics*. PhD thesis, Department of Computer Science, University of Toronto, 2008.

[47] P. Nguyen and S. A. Cook. The complexity of proving the discrete jordan curve theorem. In *Proceedings of the Twenty Second Annual IEEE Symposium on Logic in Computer Science (LICS '07)*, pages 245–256, 2007.

[48] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[49] J. Paris and L. Harrington. A mathematical incompleteness in peano arithmetic. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 1133–1142. North-Holland Publishing Company, 1977.

[50] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In A. Dold and B. Eckmann, editors, *Methods in Mathematical Logic (Proceedings Caracas 1983)*, number 1130 in Lecture Notes in Mathematics, pages 317–340. Springer Verlag, 1985.

[51] S. Perron. A propositional proof system for log space. In C.-H. L. Ong, editor, *Proceedings of the 19th international Workshop on Computer Science Logic (CSL '05)*, volume 3634 of *Lecture Notes in Computer Science*, pages 509–524. Springer Verlag, Aug. 2005.

[52] W. Pohlers. Subsystems of set theory and second order number theory. In S. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundation of Mathematics*, chapter IV, pages 209–335. Elsevier, 1998.

[53] W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierachies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, Apr. 1984.

[54] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[55] W. J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7:389–403, 1973.

[56] K. Schütte. Die unendliche Induktion in der Zahlentheorie. *Mathematische Annalen*, 122:369–389, 1951.

[57] A. Skelley. Propositional PSPACE reasoning with Boolean programs versus quantified Boolean formulas. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proceedings of the Thirty-first International Colloquium of Automata, Languages and Programming (ICALP '04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1163–1175. Springer Verlag, July 2004.

[58] A. Skelley. A third-order bounded arithmetic theory for PSPACE. In J. Marcinkowski and A. Tarlecki, editors, *Proceedings of the 18th international Workshop on Computer Science Logic (CSL '04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 340–354. Springer Verlag, Sept. 2004.

[59] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, Oct. 1976.

[60] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal of Computing*, 13(2):409–4221, 1984.

[61] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[62] W. W. Tait. Normal derivability in classical logic. In J. Barwise, editor, *The Syntax and Semantics of Infinitatry Languages*, number 72 in Lecture Notes in Mathematics, pages 204–236. Springer Verlag, 1968.

[63] G. Takeuti. Separations of theories in weak bounded arithmetic. *Annals of Pure and Applied Logic*, 71:47–67, 1995.

[64] C. B. Wilson. A measure of relativized space which is faithful with respect to depth. *Journal of Computer and System Sciences*, 36(3):303–312, June 1988.

[65] C. B. Wilson. Relativized NC. *Mathematical Systems Theory*, 20(1):13–29, 1989.

[66] D. Zambella. Notes on polynomially bounded arithmetic. *The Journal of Symbolic Logic*, 61(3):942–966, Sept. 1996.

Index