Logical Methods in Computer Science
Vol. 9(1:16)2013, pp. 1–16
www.lmcs-online.org

Submitted Mar. 16, 2012
Published Mar. 31, 2013

POLYLOGARITHMIC CUTS IN MODELS OF $\mathbf{V}^0$

SEBASTIAN MÜLLER

Faculty of Mathematics and Physics, Charles University, Prague
e-mail address: muller@karlin.mff.cuni.cz

Abstract. We study initial cuts of models of weak two-sorted Bounded Arithmetics with respect to the strength of their theories and show that these theories are stronger than the original one. More explicitly we will see that polylogarithmic cuts of models of $\mathbf{V}^0$ are models of $\mathbf{VNC}^1$ by formalizing a proof of Nepomnjascij's Theorem in such cuts. This is a strengthening of a result by Paris and Wilkie.

We can then exploit our result in Proof Complexity to observe that Frege proof systems can be sub exponentially simulated by bounded depth Frege proof systems. This result has recently been obtained by Filmus, Pitassi and Santhanam in a direct proof. As an interesting observation we also obtain an average case separation of Resolution from AC^0-Frege by applying a recent result with Tzameret.

1. Introduction

This article is on the one hand on models of weak arithmetics and on the other on proof complexity, i.e. the question of how long formal proofs of tautologies have to be in given proof systems. Therefore the introduction will consist of two parts, one for each subject.

Models of weak arithmetics, like $I\Delta_0$, have been extensively studied for several reasons. They are possibly the simplest objects whose theories bear enough strength to do a good part of mathematics in, yet they are weak enough to allow for a certain kind of constructiveness. The latter has been demonstrated over and over again by various results connecting weak arithmetic theories with complexity classes and computability. We are interested in the strength of the theory obtained by restricting our objects of reasoning to a small initial part of a given model. Since a two-sorted theory, such as $\mathbf{V}^0$, is much stronger on its number part than on its set part, it is likely that such a cut is a model of a supposedly much stronger theory. Indeed we will see in Section 3 that certain cuts of models of $\mathbf{V}^0$ are models of the provably stronger theory $\mathbf{VNC}^1$. This strengthens a result by Paris and Wilkie [18][17], who show that such cuts are models of $\mathbf{VTC}^0$. In fact they work in a more general setting and, following our argumentation, their result readily implies the sub

2012 ACM CCS: [**Theory of computation**]: Logic—Proof theory.
2010 Mathematics Subject Classification: 03B30, 03B70, 03C62, 03D15.
Key words and phrases: Proof Complexity, Bounded Arithmetic, Cuts, Subexponential Simulation.
Supported by the Marie Curie Initial Training Network in Mathematical Logic - MALOA - From MAthematical LOgic to Applications, PITN-GA-2009-238381.

DOI:10.2168/LMCS-9(1:16)2013

exponential simulation of TC^0-Frege by AC^0-Frege from Bonet, Domingo, Gavaldà, Maciel, and Pitassi [3].

Proof Complexity, on the other hand, more or less began when Cook and Reckhow [8] discovered the close connection between the lengths of formal proofs in propositional proof systems and standard complexity classes. This connection yields a possibility of dealing with the $\mathsf{coNP/NP}$ question by asking, whether there exists a propositional proof system that is polynomially bounded. We will not directly address this question here, but rather explore the relative strengths of two major proof systems, Frege and bounded depth Frege. These proof systems have been extensively studied, due to their natural appearance as classical calculi, such as Gentzen's PK, and it is well known that Frege systems are stronger than bounded depth Frege systems, as the former system has polynomial size proofs for the Pigeonhole Principle (see [5]), while the latter does not (see [14] and [19]). Lately, Filmus, Pitassi and Santhanam [10] have proved a sub exponential simulation of Frege by bounded depth Frege using a combinatoric argument. In Section 4 we will obtain the same result by an application of our result about cuts to the provability of the Reflection Principle for Frege in bounded depth Frege. Currently Cook, Ghasemloo and Nguyen [6] are working on a purely syntactical proof that gives a slightly better result with respect to the strength of the simulated proof system.

The paper is built-up as follows. In section 2 we briefly recapture some basics about Complexity Theory, Bounded Arithmetic, Proof Complexity and the various connections between them. As this is only expository it might be helpful to consult some of the references for a more detailed introduction (see [1], [7] and [12]). After that, in Section 3 we prove a formalization of Nepomnjascij's Theorem in the polylogarithmic cut of a model of $\mathbf{V}^0$. Using a standard algorithm for evaluating circuits and then applying the formalized version of Nepomnjascij's Theorem we can conclude that this cut is indeed a model of $\mathbf{VNC}^1$. Finally, in Section 4, we apply this result to prove that a version of the Bounded Reflection Principle of Frege is provable in $\mathbf{V}^0$. This, together with a standard argument linking the provability of Reflection Principles with simulation results, yields the sub exponential simulation of Frege by bounded depth Frege.

2. Preliminaries

We assume familiarity with Turing machines, circuits and standard complexity classes such as P, NP, $\mathsf{TimeSpace}(f, g)$, NC^i, AC^i and so on. See for example [1] for an introduction. We will not work a lot within these classes, but rather apply known relations between such classes and weak arithmetic theories.

We will work in a two-sorted arithmetic setting, having one sort of variables representing numbers and the second sort representing bounded sets of numbers. We identify such bounded subsets with strings. See [7] for a thorough introduction. The underlying language, denoted $\mathcal{L}_A^2$, consists of the following relation, function and constant symbols:

$$\{+, \cdot, \leq, 0, 1, |\cdot|, =_1, =_2, \in\} .$$

An $\mathcal{L}_A^2$-structure M consists of a first-sort universe U_1^M of numbers and a second-sort universe U_2^M of bounded subsets of numbers. If M is a model of the two-sorted theory $\mathbf{V}^0$ (see 2.2), then the functions $+$ and $\cdot$ are the addition and multiplication on the universe of numbers. 0 and 1 are interpreted as the appropriate elements zero and one with respect to addition and multiplication. The relation $\leq$ is an ordering relation on the first-sort universe.

The function $|\cdot|$ maps an element of the set sort to its largest element plus one (i.e. to an element of the number sort). The relation $=_1$ is interpreted as equality between numbers, $=_2$ is interpreted as equality between bounded sets of numbers. The relation $\in$ holds for a number n and a set of numbers N if and only if n is an element of N. The standard model of two-sorted Peano Arithmetic will be denoted as $\mathbb{N}_2$. It consists of a first-sort universe $U_1 = \mathbb{N}$ and a second-sort universe U_2 of all finite subsets of $\mathbb{N}$. The symbols are interpreted in the usual way.

We denote the first-sort (number) variables by lower-case letters $x, y, z, \ldots, \alpha, \beta, \gamma, \ldots$, and the second-sort (set) variables by capital letters $X, Y, Z, \ldots, A, B, \Gamma, \ldots$. In case it helps to describe the meaning of a variable we will use lower case words for first-sort and words starting with a capital letter for second-sort variables. We can build formulas in the usual way, using two sorts of quantifiers, number quantifiers and string quantifiers. A number quantifier $\exists x$ ($\forall x$) is bounded if it is of the form $\exists x(x \leq f \wedge \ldots)$ ($\forall x(x \leq f \rightarrow \ldots)$) for some number term f. A string quantifier $\exists X$ ($\forall X$) is bounded if it is of the form $\exists X(|X| \leq f \wedge \ldots)$ ($\forall X(|X| \leq f \rightarrow \ldots)$) for some number term f. A formula is bounded iff all its quantifiers are. All formulas in this paper will be bounded. A formula φ is in Σ_0^B (or Π_0^B) if it uses no string quantifiers and all number quantifiers are bounded. A formula φ is a Σ_{i+1}^B (or Π_{i+1}^B) if it is of the form $\exists X_1 \leq p(n) \ldots \exists X_m \leq p(n)\psi$ (or $\forall X_1 \leq p(n) \ldots \forall X_m \leq p(n)\psi$), where $\psi \in \Pi_i^B$ (or $\psi \in \Sigma_i^B$, respectively). If a relation or predicate can be defined by both a Σ_i^B and a Π_i^B formula, then we call it Δ_i^B definable. The *depth* of a formula is the maximal number of alternations of its logical connectives and quantifiers.

As mentioned before we will represent a bounded set of numbers N by a finite string $S_N = S_N^0 \ldots S_N^{|N|-1}$ such that $S_N^i = 1$ if and only if $i \in N$. We will abuse notation and identify bounded sets and strings, i.e. N and S_N.

Further, we will encode monotone propositional formulas inductively as binary trees in the standard way, giving a node the value 1 if it corresponds to a conjunction and the value 0, if it corresponds to a disjunction. Binary trees are encoded as strings as follows. If position x contains the value of a node $\mathfrak{n}_x$, then the value of its left successor is contained in position $2x$, while the value of its right successor is in $2x + 1$.

2.1. **Elements of Proof Complexity.** We restate some basic definitions introduced in [8].

Definition 2.1. A *propositional proof system (pps)* is a surjective polynomial-time function $P : \{0,1\}^* \longrightarrow \mathsf{TAUT}$, where TAUT is the set of propositional tautologies (in some natural encoding). A string π with $P(\pi) = \tau$ is called a *P-proof* of τ.

We can define a quasi ordering on the class of all pps as follows.

Definition 2.2. Let P, Q be propositional proof systems.

- P simulates Q (in symbols $P \geq Q$), iff there is a polynomial p, such that for all $\tau \in \mathsf{TAUT}$ there is a π_P with $P(\pi_P) = \tau$, such that for all π_Q with $Q(\pi_Q) = \tau$, $|\pi_P| \leq p(|\pi_Q|)$.
- If there is a polynomial time machine that takes Q-proofs and produces P-proofs for the same formula we say that P p-simulates Q (in symbols $P \geq_p Q$).
- If P and Q mutually (p-)simulate each other, we say that they are (p-)equivalent (in symbols $P \equiv Q$ and $P \equiv_p Q$, respectively).

In this article we will be mainly interested in bounded depth Frege systems and some of their extensions. A Frege system is a typical textbook proof system, such as Gentzen's propositional calculus PK. We will only sketch a single rule of such a system as an example and refer the interested reader to standard logic textbooks.

$$\frac{\Gamma \longrightarrow A, \Delta \qquad \Gamma, A \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}\,(\text{Cut})$$

Here, Δ and Γ are sets of formulas while A is a formula. $\Gamma \longrightarrow \Delta$ is read as "The conjunction of all formulas in Γ implies the disjunction of all formulas in Δ". The Cut Rule therefore says that, if Γ implies A or Δ, and Γ and A imply Δ, then Γ already implies Δ. The formula A is called the *Cut Formula*.

In a bounded depth Frege system the depths of all formulas in a derivation are bounded by some global constant. This is equivalent to being representable by an AC^0 circuit. Thus we also call bounded depth Frege AC^0-Frege. If the formulas are unbounded, we speak of NC^1-Frege or simply of Frege. We readily get

Fact 2.1. AC^0-Frege $\leq_p$ Frege.

A pps P is *polynomially bounded* iff there is a polynomial p such that every tautology τ has a P-proof π with $|\pi| \leq p(|\tau|)$.

We are interested in the existence of polynomially bounded pps. This is, at least in part, due to the following theorem.

Fact 2.2 ([8])**.** $\mathsf{NP} = \mathsf{coNP} \Leftrightarrow$ There exists a polynomially bounded pps.

An easier task than searching for a polynomially bounded pps might be to find some pps with sub exponential bounds to the lengths of proofs. This corresponds to the question, whether sub exponential time nondeterministic Turing machines can compute coNP-complete languages. To explore the existence of such systems we generalize Definition 2.2.

Definition 2.3. Let P, Q be propositional proof systems and F a family of increasing functions on $\mathbb{N}$.

- P F-simulates Q (in symbols $P \geq^F Q$), iff there is a function $f \in F$, such that for all $\tau \in \mathsf{TAUT}$ there is a π_P with $P(\pi_P) = \tau$, such that for all π_Q with $Q(\pi_Q) = \tau$, $|\pi_P| \leq f(|\pi_Q|)$.
- If there is an $\mathsf{Time}(F)$-machine that that takes Q-proofs and produces P-proofs for the same formula we say that P F-computably simulates Q (in symbols $P \geq_p^F Q$).
- If P and Q mutually F-(computably) simulate each other, we say that they are F-(computably) equivalent (in symbols $P \equiv^F Q$ and $P \equiv_p^F Q$, respectively).

We say a pps P *sub exponentially simulates* a pps Q iff the above F can be chosen as a class of $2^{n^{o(1)}}$ functions.

2.2. **The theory $\mathbf{V}^0$.** The base theory we will be working with is $\mathbf{V}^0$. It consists of the following axioms:

Basic 1. $x+1 \neq 0$	**Basic 2**. $x+1=y+1 \to x=y$
Basic 3. $x+0=x$	**Basic 4**. $x+(y+1)=(x+y)+1$
Basic 5. $x \cdot 0 = 0$	**Basic 6**. $x \cdot (y+1) = (x \cdot y) + x$
Basic 7. $(x \leq y \wedge y \leq x) \to x = y$	**Basic 8**. $x \leq x+y$
Basic 9. $0 \leq x$	**Basic 10**. $x \leq y \vee y \leq x$
Basic 11. $x \leq y \leftrightarrow x < y+1$	**Basic 12**. $x \neq 0 \to \exists y \leq x(y+1=x)$
L1. $X(y) \to y < \lvert X\rvert$	**L2**. $y+1 = \lvert X\rvert \to X(y)$

SE. $(\lvert X\rvert = \lvert Y\rvert \wedge \forall i \leq \lvert X\rvert\,(X(i) \leftrightarrow Y(i))) \to X = Y$

Σ^B_0**-COMP.** $\exists X \leq y \forall z < y(X(z) \leftrightarrow \varphi(z))\,, \quad$ for all $\varphi \in \Sigma^B_0$.

Here, the Axioms **Basic 1** through **Basic 12** are the usual axioms used to define Peano Arithmetic without induction (PA^-), which settle the basic properties of Addition, Multiplication, Ordering, and of the constants 0 and 1. The Axiom **L1** says that the length of a string coding a finite set is an upper bound to the size of its elements. **L2** says that $|X|$ gives the largest element of X plus 1. **SE** is the extensionality axiom for strings which states that two strings are equal if they code the same sets. Finally, Σ^B_0**-COMP** is the comprehension axiom schema for Σ^B_0-formulas (it is an axiom for each such formula) and implies the existence of all sets, which contain exactly the elements that fulfill any given Σ^B_0 property.

Fact 2.3. The theory $\mathbf{V}^0$ proves the Induction Axiom schema for Σ^B_0 formulas Φ:

$$(\Phi(0) \wedge \forall x(\Phi(x) \to \Phi(x+1))) \to \forall z\, \Phi(z).$$

When speaking about theories we will always assume that the theories are two-sorted theories as in [7].

The following is a basic notion:

Definition 2.4 (Two-sorted definability)**.** Let $\mathcal{T}$ be a theory over the language $\mathcal{L} \supseteq \mathcal{L}^2_A$ and let Φ be a set of formulas in the language $\mathcal{L}$. A number function f is Φ-definable in a theory $\mathcal{T}$ iff there is a formula $\varphi(\vec{x}, y, \vec{X})$ in Φ such that $\mathcal{T}$ proves

$$\forall \vec{x} \forall \vec{X} \exists! y \varphi(\vec{x}, y, \vec{X})$$

and it holds that

$$y = f(\vec{x}, \vec{X}) \leftrightarrow \varphi(\vec{x}, y, \vec{X}). \tag{2.1}$$

A string function F is Φ-definable in a theory $\mathcal{T}$ iff there is a formula $\varphi(\vec{x}, \vec{X}, Y)$ in Φ such that $\mathcal{T}$ proves

$$\forall \vec{x} \forall \vec{X} \exists! Y \varphi(\vec{x}, \vec{X}, Y)$$

and it holds that

$$Y = F(\vec{x}, \vec{X}) \leftrightarrow \varphi(\vec{x}, \vec{X}, Y). \tag{2.2}$$

Finally, a relation $R(\vec{x}, \vec{X})$ is Φ-definable iff there is a formula $\varphi(\vec{x}, \vec{X})$ in Φ such that it holds that

$$R(\vec{x}, \vec{X}) \leftrightarrow \varphi(\vec{x}, \vec{X}). \tag{2.3}$$

Moreover we wish to talk about sequences coded by strings or numbers. For a string X we let $X[i]$ be the ith bit of X. Assuming a tupling function $\langle \cdot, \dots, \cdot \rangle$ we can also talk of k-ary relations for any constant k. We refer to $X[\langle i_0, \dots, i_{k-1} \rangle]$, to say that the objects $i_0, \dots, i_{k-1}$ are in the relation X (which is equivalent to saying that the predicate X holds for the number $\langle i_0, \dots, i_{k-1} \rangle$, i.e. that the X contains that number as an element). For the sake of simplicity we also refer to $X[\langle i_0, \dots, i_k \rangle]$ by $X[i_0, \dots, i_k]$.

Using k-ary relations we can also encode sequences of bounded numbers $x_0, \dots, x_m$ by $x_i = X[\langle i, 0 \rangle] X[\langle i, 1 \rangle] \dots X[\langle i, k \rangle]$ in binary. Matrices and so on can obviously be formalized in the same way.

Given a string $X[\langle x_1, \dots, x_k \rangle]$ representing a k-ary relation, we denote the $k - \ell$-ary substring with parameters $a_{i_1}, \dots, a_{i_\ell}$ by $X[\langle \cdot, \dots, \cdot, a_{i_1}, \cdot, \dots, a_{i_\ell}, \cdot, \dots, \cdot \rangle]$. For example we refer to the element a_{ij} of a given matrix $A[\langle x_1, x_2, x_3 \rangle]$ as $A[\langle i, j, \cdot \rangle]$, a string representing a_{ij} in binary. Observe that this substring can be Σ^B_0 defined in $\mathbf{V}^0$.

Given a number x we denote by $\langle x \rangle_j$ the jth number in the sequence encoded by x. To do this we assume a fixed Σ^B_0 definable encoding of numbers that is injective. The sequence itself will be addressed as $\langle x \rangle$. As above, we can also talk about matrices, etc. in this way, i.e. read such a sequence as a sequence of k-tuples.

We want to identify strings of short length with sequences of numbers. Thus, given a string X of length $O(n)$ we can Σ^B_0-define (in $\mathbf{V}^0$) a number $x \leq 2^{O(n)}$ that codes a sequence $\langle x \rangle$, such that $X[i] = \langle x \rangle_i$ for all $i < |X|$ and vice versa. We will use $\langle x \rangle \approx X$ and $\langle x \rangle_i \approx X[i]$ to denote the above identification. Observe that n has to be very small in order to be able to do the above in $\mathbf{V}^0$.

2.2.1. *Computations in models of* $\mathbf{V}^0$. Given a polynomially bounded Turing machine A in a binary encoding, we can Σ^B_1 define a predicate $\mathsf{ACC}_A(X)$, that states that X is accepted by A. This can readily be observed, since, provided some machine A, there is a constant number of states $\sigma_1, \dots, \sigma_k$ and the whole computation can be written into a matrix W of polynomial size. That W is indeed a correct computation can then be easily checked, because the computations are only local.

More precisely let $A = \langle \sigma_1, \dots, \sigma_k; \delta \rangle$ be given, where the σ_i are different states, with σ_1 being the initial state and σ_k being the accepting state and δ is the transition function with domain $\{\sigma_1, \dots, \sigma_k\} \times \{0, 1\}$ and range $\{\sigma_1, \dots, \sigma_k\} \times \{0, 1\} \times \{\leftarrow, \downarrow, \rightarrow\}$, which describes what the machine does. I.e. if $\delta(a, b) = (c, d, e)$, then if the machine is in state a and reads b, it replaces b by d, goes into state c and moves one position on the tape in the direction e. For our formalization we will assume a function $\delta : \mathbb{N}^2 \to \mathbb{N}$ and interpret it in the following way, $\delta(\sigma_a, b) = (\langle \delta(\sigma_a, b) \rangle_1, \langle \delta(\sigma_a, b) \rangle_2, \langle \delta(\sigma_a, b) \rangle_3)$, where we identify $\downarrow = 0, \leftarrow = 1, \rightarrow = 2$.

Let the polynomial p bound the running time of A, then we can formalize $\mathsf{ACC}_A(X)$ as follows

$$\begin{aligned}
&\exists W \leq (p(|X|)^2 \cdot \log(k)^2)\forall i, i' \leq p(|X|)\forall 0 < \alpha \leq k(\\
&\quad i < |X| \rightarrow (\langle W[\langle 0, i, \cdot\rangle]\rangle_1 = X[i] \wedge i > 0 \rightarrow \langle W[\langle 0, i, \cdot\rangle]\rangle_2 = 0 \wedge \langle W[\langle 0, 0, \cdot\rangle]\rangle_2 = 1)\wedge\\
&\quad i \geq |X| \rightarrow (\langle W[\langle 0, i, \cdot\rangle]\rangle_1 = 0 \wedge \langle W[\langle 0, i, \cdot\rangle]\rangle_2 = 0)\wedge\\
&\quad \langle W[\langle j, i, \cdot\rangle]\rangle_2 = 0 \rightarrow (\langle W[\langle j+1, i, \cdot\rangle]\rangle_1 = \langle W[\langle j, i, \cdot\rangle]\rangle_1)\wedge\\
&\quad \langle W[\langle j, i, \cdot\rangle]\rangle_2 = \alpha \rightarrow (\langle W[\langle j+1, i, \cdot\rangle]\rangle_1 = \langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_2\wedge\\
&\quad\ (\langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_3 = 0 \rightarrow \langle W[\langle j+1, i, \cdot\rangle]\rangle_2 = \langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_1)\wedge\\
&\quad\ (\langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_3 = 1 \rightarrow \langle W[\langle j+1, i \dot{-} 1, \cdot\rangle]\rangle_2 = \langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_1)\wedge\\
&\quad\ (\langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_3 = 2 \rightarrow \langle W[\langle j+1, i+1, \cdot\rangle]\rangle_2 = \langle \delta(\alpha, \langle W[\langle j, i, \cdot\rangle]\rangle_1)\rangle_1))\wedge\\
&\quad (i \neq i' \rightarrow (\langle W[\langle j, i, \cdot\rangle]\rangle_2 > 0 \rightarrow \langle W[\langle j, i', \cdot\rangle]\rangle_2 = 0)) \wedge W[\langle p(|X|), i, \cdot\rangle]\rangle_2 = k).
\end{aligned} \tag{2.4}$$

Thus, in plain English, $\mathsf{ACC}_A(X)$ says that there exists a matrix W of pairs of numbers that witnesses an accepting computation of A. Here, $\langle W[\langle i, j, \cdot\rangle]\rangle$ is supposed to code the jth cell on the Turing machine's tape after i steps of computations on input X. As noted above, $\langle W[\langle i, j, \cdot\rangle]\rangle_1$ is a binary number, which is the value of the cell, $\langle W[\langle i, j, \cdot\rangle]\rangle_2$ is a number coding the state the machine is in iff the pointer is on that cell.

The second and third line of the definition say that the tape in the initial step contains X padded with zeroes in the end to get the proper length ($p(|X|)$) and that the read/write head is in its starting state and position. The fourth line says that if the read/write head is not on cell i, then nothing happens to the content of cell i. The fifth line says that the content of the cell, where the read/write head is in step j, is changed according to δ. The next three lines tell us where the read/write head moves. The last line says that there is at most one position on the tape where the read/write head may be at any step and that the state after the last step is accepting.

We also define a Σ_1^B-predicate $\mathsf{REACH}_A(Y, Y')$ that says that A reaches configuration Y' from configuration Y in at most $p(|Y|)$ many steps. This is essentially the same predicate as ACC, with the constraints on the initial and accepting state lifted and instead a constraint added that the first line of computation is Y and the last is Y'. We omit the details as it does not severely differ from the above definition of ACC.

2.3. **Extensions of $\mathbf{V}^0$.** The Theory $\mathbf{V}^0$ serves as our base theory to describe complexity classes by arithmetical means.

The problem, whether a given monotone formula φ of size ℓ and depth $\lceil \log(\ell) \rceil$ is satisfiable under a given assignment I is AC^0-complete for NC^1. Therefore Cook and Nguyen ([7]) define the class $\mathbf{VNC}^1$ as $\mathbf{V}^0$ augmented by the axiom $MFV \equiv \exists Y \leq 2a+1.\delta_{MFV}(a, G, I, Y)$, where

$$\begin{aligned}
\delta_{MFV}(a, G, I, Y) \equiv \forall x < a((Y(x+a) \leftrightarrow I(x)) \wedge Y(0)\wedge\\
0 < x \rightarrow (Y(x) \leftrightarrow ((G(x) \wedge Y(2x) \wedge Y(2x+1))\vee\\
(\neg G(x) \wedge (Y(2x) \vee Y(2x+1)))))).
\end{aligned}$$

So, MFV states that there is an evaluation Y of the monotone formula represented by G under the assignment given by I of length at most $2a+1$. More specifically, G is a

tree-encoding of the formula, where $G(x)$ is true, if node x is $\wedge$ and false, if x is $\vee$. The evaluation Y takes the value of the variables given by I and then evaluates the formula in a bottom-up fashion using a standard tree encoding. Thus, the value of the formula can be read at $Y(1)$.

It is interesting to observe that MFV does not hold in $\mathbf{V}^0$. This is, since an application of the Witnessing Theorem for $\mathbf{V}^0$ to a proof of MFV would yield an AC^0-definition of satisfaction for monotone NC^1 circuits. This implies that monotone $\mathsf{NC}^1 \subseteq \mathsf{AC}^0$, which is known to be false.

2.4. **Relation between Arithmetic Theories and Proof Systems.** In this section we will remind the reader of a connection between the Theory $\mathbf{V}^0$ and some of its extensions and certain propositional proof systems (see also [7][12]).

Definition 2.5. The following predicates will be subsequently used. They are definable with respect to $\mathbf{V}^0$ (see [12]).

- $\mathsf{Fla}(X)$ is a Σ_0^B formula that says that the the string X codes a formula.
- $Z \models X$ is the Δ_1^B definable property that the truth assignment Z satisfies the formula X.
- $\mathsf{Taut}(X)$ is the Π_1^B formula $\mathsf{Fla}(X) \wedge \forall Z \leq t(|X|) Z \models X$, where t is an upper bound to the number of variables in formulas coded by strings of length $|X|$.
- $\mathsf{Prf}_{F_d}(\Pi, A)$ is a Σ_0^B definable predicate meaning Π is a depth d Frege proof for A.
- $\mathsf{Prf}_{F}(\Pi, A)$ is a Σ_0^B definable predicate meaning Π is a Frege proof for A.

The following holds

Fact 2.4 (see [7])**.** The Theory $\mathbf{V}^0$ proves that AC^0-Frege is sound, i.e. for every d

$$\forall A \forall \Pi \mathsf{Prf}_{F_d}(\Pi, A) \rightarrow \mathsf{Taut}(A).$$

Fact 2.5 (see [7])**.** The Theory $\mathbf{VNC}^1$ proves that Frege is sound, i.e.

$$\forall A \forall \Pi \mathsf{Prf}_{F}(\Pi, A) \rightarrow \mathsf{Taut}(A).$$

On the other hand, provability of the universal closure of Σ_0^B formulas in $\mathbf{V}^0$ and $\mathbf{VNC}^1$ implies the existence of polynomial size proofs of their propositional translations in AC^0-Frege and Frege, respectively.

The propositional translation $[\![\varphi(\bar{x}, \bar{X})]\!]_{\bar{m},\bar{n}}$ of a Σ_0^B formula $\varphi(\bar{x}, \bar{X})$ is a family of propositional formulas built up inductively (on the logical depth) as follows. If φ is atomic and does not contain second sort variables, we evaluate φ in $\mathbb{N}_2$, if it contains second sort variables, we have to introduce propositional variables. If φ is a boolean combination of formulas ψ_i of lower depth, the translation is simply the same boolean combination of the translations of the ψ_i. If φ is $\exists \psi$ or $\forall \psi$ we translate it to the disjunction or conjunction of the translations, respectively. For a proper definition see [7].

Fact 2.6. There exists a polynomial p such that for all Σ_0^B formulas $\varphi(\bar{x}, \bar{X})$ the following holds

- If $\mathbf{V}^0 \vdash \forall \bar{X} \forall \bar{x} \varphi(\bar{x}, \bar{X})$, then there exist bounded depth Frege proofs of all $[\![\varphi]\!]_{\bar{m},\bar{n}}$ of length at most $p(\max(\bar{m}, \bar{n}))$, for any $\bar{m}, \bar{n}$.
- If $\mathbf{VNC}^1 \vdash \forall \bar{X} \forall \bar{x} \varphi(\bar{x}, \bar{X})$, then there exist Frege proofs of all $[\![\varphi]\!]_{\bar{m},\bar{n}}$ of length at most $p(\max(\bar{m}, \bar{n}))$, for any $\bar{m}, \bar{n}$.

These proofs are effective in the sense that for any such φ there exists a polynomial-time computable function F_φ that maps any tuple $(\bar{m}, \bar{n})$ to the above proofs of $[\![\varphi]\!]_{\bar{m},\bar{n}}$.

Facts 2.4 and 2.5 are examples of general principles, the so called *Reflection Principles*, which are defined as follows.

Definition 2.6 (Reflection Principle)**.** Let P be a pps. Then the *Reflection Principle* for P, Ref_P, is the $\forall\Delta_1^B$-formula (w.r.t. $\mathbf{V}^0$)

$$\forall\Pi\forall X\forall Z((\mathsf{Fla}(X) \wedge \mathsf{Prf}_P(\Pi, X)) \to (Z \vDash X)),$$

where Prf_P is a Δ_1^B-predicate formalizing P-proofs.

Reflection Principles condense the strength of propositional proof systems. In what follows we will summarize some such results for the proof systems and theories used here. A detailed exposition can be found in [7], chapter X, or in [12], chapter 9.3.

Theorem 2.1. If $\mathbf{V}^0 \vdash \mathsf{Ref}_F$ then bounded depth Frege p-simulates Frege.

We will only give a brief sketch of the proof here and leave out the technical details.

Sketch. Let φ be a formula and π_φ a Frege proof of φ which is witnessed by a Turing machine T_F (cf Def 2.1). Since $\mathbf{V}^0$ proves Ref_F, by Facts 2.4 and 2.6 we have polynomial size proofs of its translations $[\![\mathsf{Ref}_F]\!]$ in bounded depth Frege. Bounded depth Frege itself, however, is strong enough to verify that a proper encoding of the computation of T_F on input (π_φ, φ) is correct. Thus it can verify that π_φ is a Frege-proof and, using the translation of the Reflection Principle and the Cut rule, conclude $[\![\mathsf{Taut}(\varphi)]\!]$. From this φ follows, cf. [12] Lemma 9.3.7. □

Given a term t and a variable x, we can also introduce the t-bounded version of the Reflection Principle for some given pps P, $\mathsf{Ref}_P(t(x))$ that claims soundness only for t-bounded proofs.

Definition 2.7 (Bounded Reflection)**.** Let t be a $\mathcal{L}_A^2$-term, x a first-sort variable and P a pps. Then the *Bounded Reflection Principle* $\mathsf{Ref}_P(t(x))$ is the formula

$$\forall\Pi \leq t(x)\forall X \leq t(x)\forall Z \leq t(x)((\mathsf{Fla}(X) \wedge \mathsf{Prf}_P(\Pi, X)) \to (Z \vDash X)).$$

We can now generalize Theorem 2.1 in the following way.

Theorem 2.2. Let t be a $\mathcal{L}_A^2$-term and x a number variable. If $t(x) < x$ for x large enough and if $\mathbf{V}^0 \vdash \forall x\mathsf{Ref}_F(t(x))$ then for every propositional formula φ with a Frege proof of length $t(x)$ there is a bounded depth Frege proof of φ of length $x^{O(1)}$. This proof can be efficiently constructed.

Proof. The proof is the same as that of Theorem 2.1. Using the Bounded Reflection Principle we can encode Frege proofs of length $t(x)$ as bounded depth Frege proofs of length $x^{O(1)}$. □

As a corollary we get

Corollary 2.3. If $\mathbf{V}^0 \vdash \mathsf{Ref}_F(|x|^k)$ for all $k \in \mathbb{N}$, then bounded depth Frege sub exponentially simulates Frege: For all $D > 1, \delta > 0$ exists $d \geq D$, such that the existence of a Frege proof of length m of a depth D formula implies the existence of a depth d Frege proof of length at most 2^{m^δ}.

3. Polylogarithmic Cuts of Models of $\mathbf{V}^0$ are Models of $\mathbf{VNC}^1$.

We will first introduce the notion of a cut $\mathcal{I}$ of a given two-sorted arithmetic model $\mathcal{M}$. This model theoretic approach provides a very good insight on what actually happens semantically with the small elements of arithmetical models.

Definition 3.1 (Cut)**.** Let T be a two-sorted arithmetic theory and

$$N = \{N_1, N_2, +^N, \cdot^N, \leq^N, 0^N, 1^N, |\cdot|^N, =_1^N, =_2^N, \in^N\}$$

a model of T. A *cut*

$$M = \{M_1, M_2, +^M, \cdot^M, \leq^M, 0^M, 1^M, |\cdot|^M, =_1^M, =_2^M, \in^M\}$$

of N is any substructure such that

- $M_1 \subseteq N_1$, $M_2 \subseteq N_2$,
- $0^M = 0^N$, $1^M = 1^N$,
- M_1 is closed under $+^N, \cdot^N$ and downwards with respect to $\leq^N$,
- $M_2 = \{X \in N_2 \mid X \subseteq M_1\}$, and
- $\circ^M$ is the restriction of $\circ^N$ to M_1 and M_2 for all relation and function symbols $\circ \in \mathcal{L}_A^2$.

We call this cut the *Polylogarithmic Cut* iff

$$x \in M_1 \Leftrightarrow \exists a \in N_1, k \in \mathbb{N}\ x \leq |a|^k .$$

To examine the strength of the theory of such cuts of models of $\mathbf{V}^0$, we will show that a formal connection between efficient computability and Σ_0^B-definability holds. This stands in contrast to general bounded subsets, where the connection is presumably only with respect to Σ_1^B-definability via the predicate ACC (see (2.4) on page 7). The intended theorem is a formalization of Nepomnjascij's Theorem [16] (see also [12] pg.20). We will sketch the original proof before starting the formalization.

Theorem 3.1 (Nepomnjascij [16])**.** Let $c \in \mathbb{N}$ and $0 < \epsilon < 1$ be constants. Then if the language $L \in \mathbf{TimeSpace}(n^c, n^\epsilon)$, the relation $x \in L$ is definable by a Σ_0^B-formula over $\mathbb{N}$.

Proof. We will prove the theorem by induction on k for $L \in \mathbf{TimeSpace}(n^{k\cdot(1-\epsilon)}, n^\epsilon)$.

Let $k = 1$ and $L \in \mathbf{TimeSpace}(n^{k\cdot(1-\epsilon)}, n^\epsilon)$. For any $x \leq 2^n$ the whole computation can be coded by a number y of size $2^{O(n)}$. The existence of such a computation gives the desired Σ_0^B-definability.

For $k > 1$ we write a sequence $y_0, y_1, \ldots, y_{n^{1-\epsilon}}$ of intermediate results coding the computation, where y_0 codes the starting configuration on input x, such that we can verify that y_{i+1} is computable from y_i in $\mathbf{TimeSpace}(n^{(k-1)\cdot(1-\epsilon)}, n^\epsilon)$. By assumption there exists a Σ_0^B-formula reach^{k-1} such that $\mathsf{reach}^{k-1}(y_i, y_{i+1})$ holds iff y_{i+1} is computed from y_i. Additionally, the whole sequence has length $O(n)$ and so we can write the sequence of intermediate results y_i as a number y of length $O(n)$. Now, the Σ_0^B-definition of $x \in L$ is simply

$$\begin{aligned}\exists y \leq 2^{O(n)} \forall i \leq n^{1-\epsilon} &\mathsf{reach}^{k-1}(\langle y\rangle_i, \langle y\rangle_{i+1}) \\ &\wedge \langle y\rangle_0 \text{ encodes the starting configuration of } A \text{ on input } x \\ &\wedge \langle y\rangle_{n^{1-\epsilon}} \text{ is in an accepting state.}\end{aligned}$$

□

We will now formalize this result in $\mathbf{V}^0$ as follows

Theorem 3.2. Let $N \vDash \mathbf{V}^0$. Let $m = |a|$ for some $a \in N_1$ and let $c, k \in \mathbb{N}$ and $\epsilon < \frac{1}{k}$. If $L \in \mathsf{TimeSpace}(m^c, m^\epsilon)$ (for strings of length m) is computed by Turing machine A, then there exists a Σ_0^B definition in N of the Σ_1^B-predicate ACC_A on the interval $[0, m^k]$. I.e. any $Y \in L$, bounded by m^k is Σ_0^B-definable in N and therefore exists in the polylogarithmic cut of N.

The following version of the proof stems from a discussion with Stephen Cook and Neil Thapen during the SAS programme in Cambridge. It is more explicit than the original one and clarifies the argument.

Proof. We will inductively on d define a Σ_0^B relation $\mathsf{reach}_A^d(I, p_1, p_2, \mathsf{cell}, \mathsf{comp})$ that states that the p_2th cell of the work tape of A, starting on configuration I and computing for $p_1 \cdot m^{d\frac{1-k\epsilon}{k}}$ steps via the computation comp is cell. We will bound the quantifiers in such a way that we can conclude that both variables can be of the number sort. As d depends only on A and k we will be doing this induction outside of the theory to construct d many formulas. We will then prove the above mentioned properties of reach_A^d by Σ_0^B induction on p_1.

Keep in mind that a cell is given as a pair $\langle bit, state\rangle$, where bit is the actual value of the cell and $state$ is a number > 0 coding the state the Turing machine is in iff the pointer is on that cell and 0 otherwise. As before the transition function is denoted by δ. We let I be a string coding the input at the start of the computation. That is, I is a sequence of length $len(I) \leq m^k$, such that $I[1,1] = 1$, $I[j,1] = 0$ for all $j > 1$ and $I[j,0]$ is the jth input bit. We let $\langle\mathsf{comp}\rangle$ be a sequence encoding the computation of A, such that $\langle\mathsf{comp}\rangle_{\langle j,j',1\rangle}$ is the state, the machine is in after j steps (0 denotes that the read/write head is not on cell j', while a greater number gives the state and witnesses that the read/write head is on cell j'). $\langle\mathsf{comp}\rangle_{\langle j,j',0\rangle}$ is the value of cell j' after j steps of the computation. Observe that this also implies that the computation can be encoded as a number, that is, it has to be very short. This is straight forward from the quantifier bounds.

We can now define

$$
\begin{aligned}
&\mathsf{reach}_A^0(I, p_1, p_2, \mathsf{cell}, \mathsf{comp})\\
&\equiv (\forall j' < \lceil len(I)^\epsilon\rceil, \langle\mathsf{comp}\rangle_{\langle 1,j',0\rangle} \approx I[j',0] \wedge \langle\mathsf{comp}\rangle_{\langle 1,j',1\rangle} \approx I[j',1]) \wedge\\
&\quad [\forall j < \lceil len(I)^{\frac{1-k\epsilon}{k}}\rceil, j' < \lceil len(I)^\epsilon\rceil, \alpha < |A|\\
&\qquad (\langle\mathsf{comp}\rangle_{\langle j,j',1\rangle} = 0 \rightarrow (\langle\mathsf{comp}\rangle_{\langle j,j',0\rangle} = \langle\mathsf{comp}\rangle_{\langle j+1,j',0\rangle})) \wedge\\
&\qquad (\langle\mathsf{comp}\rangle_{\langle j,j',1\rangle} = \alpha \rightarrow (\\
&\qquad\quad (\langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_3 = 0 \rightarrow (\langle\mathsf{comp}\rangle_{\langle j+1,j',1\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_1 \wedge\\
&\qquad\qquad\qquad \langle\mathsf{comp}\rangle_{\langle j+1,j',0\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_2)) \wedge\\
&\qquad\quad (\langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_3 = 1 \rightarrow \langle\mathsf{comp}\rangle_{\langle j+1,j' \dot{-} 1,1\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_1 \wedge\\
&\qquad\qquad\qquad \langle\mathsf{comp}\rangle_{\langle j+1,j',0\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_2)) \wedge\\
&\qquad\quad (\langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_3 = 2 \rightarrow \langle\mathsf{comp}\rangle_{\langle j+1,j'+1,1\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_1 \wedge\\
&\qquad\qquad\qquad \langle\mathsf{comp}\rangle_{\langle j+1,j',0\rangle} = \langle\delta(\alpha, \langle\mathsf{comp}\rangle_{\langle j,j',0\rangle})\rangle_2)))) \wedge\\
&\qquad \forall \ell \neq \ell' < \lceil len(I)^\epsilon\rceil (\langle\mathsf{comp}\rangle_{\langle j,\ell,1\rangle} > 0 \rightarrow \langle\mathsf{comp}\rangle_{\langle j,\ell',1\rangle} = 0)] \wedge\\
&\quad (\langle\mathsf{cell}\rangle_1 = \langle\mathsf{comp}\rangle_{\langle p_1,p_2,0\rangle}) \wedge (\langle\mathsf{cell}\rangle_2 = \langle\mathsf{comp}\rangle_{\langle p_1,p_2,1\rangle}).
\end{aligned}
$$

It is straightforward to prove by induction on the number of lines in comp that comp is uniquely defined by reach^0_A. We let

$$\mathsf{Reach}^0_A(I, p_1, p_2, \mathsf{cell}) =_{def} \exists \mathsf{comp} < q(|I|)\ \mathsf{reach}^0_A(I, p_1, p_2, \mathsf{cell}, \mathsf{comp}),$$

where q is some polynomial depending on the encoding. Here it is vital that q can be defined such that $q(|I|)$ is a number in N. This is possible due to the quantifier bounds we used when defining reach^0_A. Thus, Reach^0_A is defined by a Σ^B_0 formula.

Informally Reach^0_A formalizes that there is a computation

$$\begin{array}{cccc}
\langle \mathsf{comp}\rangle_{\langle 1,1,\cdot\rangle} & \langle \mathsf{comp}\rangle_{\langle 1,2,\cdot\rangle} & \cdots & \langle \mathsf{comp}\rangle_{\langle 1,(m^k)^\epsilon,\cdot\rangle} \\
\langle \mathsf{comp}\rangle_{\langle 2,1,\cdot\rangle} & \langle \mathsf{comp}\rangle_{\langle 2,2,\cdot\rangle} & \cdots & \langle \mathsf{comp}\rangle_{\langle 2,(m^k)^\epsilon,\cdot\rangle} \\
\vdots & \ddots & & \vdots \\
\vdots & & \ddots & \vdots \\
\langle \mathsf{comp}\rangle_{\langle (m^k)^{\frac{1-k\epsilon}{k}},1,\cdot\rangle} & \langle \mathsf{comp}\rangle_{\langle (m^k)^{\frac{1-k\epsilon}{k}},2,\cdot\rangle} & \cdots & \langle \mathsf{comp}\rangle_{\langle (m^k)^{\frac{1-k\epsilon}{k}},(m^k)^\epsilon,\cdot\rangle}
\end{array}$$

that is correct in the sense that we can verify that we get from line to line via the transition function of A and gives the appropriate values of the cell at (p_1, p_2). Observe that the size of the whole computation as presented above is linear in m, i.e. that it can be coded as a number in N.

We will now proceed by inductively defining reach^d_A and Reach^d_A. Assume that $\mathsf{reach}^{d\dot{-}1}_A$ has already been defined by a Σ^B_0 formula over N. We then let

$$\begin{aligned}
&\mathsf{reach}^d_A(I, p_1, p_2, \mathsf{cell}, \mathsf{comp}) \\
&\quad\equiv (\forall j' < \lceil len(I)^\epsilon \rceil, \langle \mathsf{comp}\rangle_{\langle 1,j',0\rangle} \approx I[j', 0] \wedge \langle \mathsf{comp}\rangle_{\langle 1,j',1\rangle} \approx I[j', 1]) \wedge \\
&\qquad (\forall j < \lceil len(I)^{\frac{1-k\epsilon}{k}} \rceil \exists \mathsf{comp}' < q(len(I)) \forall j' < \lceil len(I)^\epsilon \rceil \exists \mathsf{cell}' < |A|\, \forall j'' < |A| \\
&\qquad (\langle \mathsf{comp}\rangle_{\langle j+1,j',j''\rangle} \leftrightarrow \langle \mathsf{cell}'\rangle_{j''}) \wedge \\
&\qquad \mathsf{reach}^{d\dot{-}1}_A(\langle \mathsf{comp}\rangle_{\langle j,\cdot,\cdot\rangle}, m^{\frac{1-k\epsilon}{k}}, j', \mathsf{cell}', \mathsf{comp}')) \wedge \\
&\qquad (\langle \mathsf{cell}\rangle_1 \leftrightarrow \langle \mathsf{comp}\rangle_{\langle p_1,p_2,0\rangle}) \wedge (\langle \mathsf{cell}\rangle_2 = \langle \mathsf{comp}\rangle_{\langle p_1,p_2,1\rangle}).
\end{aligned}$$

Again, we can prove uniqueness of the computation by induction on the number of its lines and let

$$\mathsf{Reach}^d_A(i, p_1, p_2, \mathsf{cell}) =_{def} \exists \mathsf{comp} < q(|I|)\ \mathsf{reach}^d_A(i, p_1, p_2, \mathsf{cell}, \mathsf{comp}).$$

That this is a Σ^B_0 definition follows by induction and the same argument as for Reach^0_A. Here, the predicate $\mathsf{Reach}^{d-1}_A(i, p_1, p_2, \mathsf{cell})$ takes the role of the transition function in witnessing that each line follows from the preceding one. The total size again is linear in m.

We now can give a Σ^B_0 definition of the predicate $W[\langle i, j, \cdot\rangle]$ coding the computation as in ACC_A on input X of length m^k. We let $W[\langle i, j, \cdot\rangle] = \mathsf{cell} \equiv$

$$\begin{aligned}
\exists r_0, \ldots, r_d < |X|^{\frac{1-k\epsilon}{k}}, con_1, \ldots, con_d < p(|X|^\epsilon) \forall z_1, \ldots, z_d < |X|^\epsilon\, \exists \mathsf{cell}_1, \ldots, \mathsf{cell}_d < |A| \\
(i = \sum_{\ell=0}^{d} r_\ell \cdot |X|^{\ell\frac{1-k\epsilon}{k}} \wedge \mathsf{Reach}^d_A(\tilde{X}, r_d, z_d, \mathsf{cell}_d) \wedge \langle con_d\rangle_{z_d} = \mathsf{cell}_d \\
\wedge \mathsf{Reach}^{d-1}_A(con_d, r_{d-1}, z_{d-1}, \mathsf{cell}_d) \wedge \langle con_{d-1}\rangle_{z_{d-1}} = \mathsf{cell}_{d-1} \\
\vdots \\
\wedge \mathsf{Reach}^0_A(con_1, r_0, j, \mathsf{cell})),
\end{aligned}$$

where p is a polynomial depending on the encoding and $\tilde{X}$ is the starting configuration of A on input X.

Informally the above formula says that we compute the configurations of A by using the predicates Reach^d_A through Reach^0_A. That is, after the application of Reach^d_A (i.e. after making the biggest steps) we have reached configuration con_d, which we plug into Reach^{d-1}_A to get configuration con_{d-1} and so on. It remains to show that this definition of $W[\langle i, j, \cdot\rangle]$ coincides with the real one, i.e. that $W[\langle i+1, \cdot, \cdot\rangle]$ follows from an application of the transition function of A from $W[\langle i, \cdot, \cdot\rangle]$.

We will prove this inductively, depending on i. Again let r_ℓ be such that $i = \sum_\ell r_\ell \cdot |X|^{\ell\frac{1-k\epsilon}{k}}$. If $i < |X|^{\frac{1-k\epsilon}{k}}$ the assumption follows straightforwardly from the definition of Reach^0_A. Now for bigger i. If the r_0, given as above, is bigger then 0 then again the assumption follows from the definition of reach^0_A. Now let $\ell > 0$ be the first index with $r_\ell > 0$. We the have to argue that $\mathsf{reach}^{\ell'-1}_A$ has the desired property. This, however, follows straightforward if we can verify this assertion for $\mathsf{reach}^{\ell'-1}_A$. Observe that d is a constant depending only on A and k, so we need to make this argument only a constant number of steps to reach reach^0_A, where we know that the assertion holds.

Since we can code the whole computation as a Σ^B_0-formula (in N), we can easily deduce a Σ^B_0-definition of the related set by simply stating that the computation accepts (i.e. that in the last line of the computation the state is accepting). This concludes the proof. □

We can now prove our main result.

Theorem 3.3. Let $N \vDash \mathbf{V}^0$ and $M \subseteq N$ be the polylogarithmic cut. Then $M \vDash \mathbf{VNC}^1$.

Proof. We have to prove that for all strings $G_\varphi \in M_2$, representing a formula φ as a tree and assignments $I \in M_2$ to its variables (i.e. leafs in the tree representation) a string Y exists in M_2 that contains all values of φ's subformulas as in the definition of MFV in Section 2.3 and satisfies the inductive conditions of MFV.

However, by Σ^B_0-comprehension and the formalized Nepomnjascij's Theorem it suffices to describe an algorithm that computes, for given G_φ and I, whether i belongs to Y in $\mathsf{TimeSpace}(|G_\varphi|^k, |G_\varphi|^\epsilon)$ for some $k \in \mathbb{N}$ and $\epsilon < 1$.

The following is a recursive algorithm computing the value of $Y[i]$, given $G := G_\varphi, I$ and i.

```
NodeValue(G,I,i)
```

- `boolean left; boolean right;`
- `If i>2·|G|`
  - `Output (0); End;`
- `Else If i>|G|`
  - `Output (I[i-|G|]); End;`
- `Else If G[i]=1`
  - `left := NodeVal(G,I,2i);`
  - `right := NodeVal(G,I,2i+1);`
  - `Output (left AND right); End;`
- `Else If G[i]=0`
  - `left := NodeVal(G,I,2i);`
  - `right := NodeVal(G,I,2i+1);`
  - `Output (left OR right); End;`
- `Else`
  - `Output (0); End;`

Observe that the algorithm at any given point only stores a constant amount of data per level of the tree G and therefore uses only $O(\log(|G|))$ space. The number steps the algorithm makes is clearly polynomial in the size of G. Therefore by Theorem 3.2, for every monotone formula φ, representable as a tree in M, we get a Σ_0^B formula eval_φ, such that $\mathsf{eval}_\varphi(i, I) \equiv Y[i]$. Observe that eval_φ depends on the size of φ and on its logical depth, as the first is essentially the size of the input for the machine, that $eval_\varphi$ codes, while the latter determines the longest iterations in the recursive algorithm. Applying the Comprehension Schema in $\mathbf{V}^0$, i.e. in N, this verifies the existence of a Y as in MFV for all formulas represented by trees in M.Therefore MFV holds in M and so $M \vDash \mathbf{VNC}^1$. □

4. Implications for Proof Complexity

We now wish to apply the above results to propositional proof systems. More precisely we wish to show that theories of small cuts of a model of a given theory $\mathcal{T}$ correspond to stronger proof systems than $\mathcal{T}$ does. An elegant way of showing such a statement is via the *Reflection Principles* of the given proof systems, i.e. the statement that the proof system is correct, as explained in Section 2.4. With their help we can conclude the following recent result of Filmus, Pitassi and Santhanam [10].

Theorem 4.1 ([10])**.** Every Frege system is sub exponentially simulated by AC^0-Frege systems.

Proof. By Theorem 2.2 we have to prove the polylogarithmically bounded Reflection Principle for Frege in $\mathbf{V}^0$. This, by Theorem 3.3 however, corresponds to proving the Reflection Principle for Frege in $\mathbf{VNC}^1$, which holds by Fact 2.5. It also follows from Theorem 2.2 that this proof can be efficiently computed from the Frege proof. □

Another, related, application is in the separation of propositional proof systems. In [15] we proved the following.

Proposition 4.2. For almost every random 3CNF A with n variables and $m = c \cdot n^{1,4}$ clauses, where c is a large constant, $\neg A$ has polynomially bounded TC^0-Frege proofs.

On the other hand it is well known (see for example [9]) that such formulas have no subexponential refutation in Resolution. Thus, this yields an average case separation between Resolution and TC^0-Frege. We can now extend this result to an average case separation between Resolution and AC^0-Frege as follows.

Theorem 4.3. For almost every random 3CNF A with n variables and $m = c \cdot n^{1,4}$ clauses, where c is a large constant, $\neg A$ has subexponentially bounded AC^0-Frege proofs.

Proof. By Theorem 3.3 the polylogarithmic Cut of any $\mathbf{V}^0$-model is a model of $\mathbf{VNC}^1$, therefore also of $\mathbf{VTC}^0$. This yields, as in our proof of Theorem 4.1, that AC^0-Frege subexponentially simulates TC^0-Frege. The result now follows from Proposition 4.2. □

5. Conclusion and Discussion

As we have seen cuts of models of weak arithmetics constitute an appropriate way for reasoning about super-polynomial simulations between proof systems. An advantage in comparison to syntactic arguments is the possible applicability of results in Model Theory and a more uniform treatment. This can readily be observed as with our argument, e.g. the work of Paris and Wilkie [17][18] immediately imply the simulation results from Bonet et al. [3].

This leads to interesting possibilities for further research, especially towards the weak automatizability of weak propositional proof systems such as Resolution. The underlying theory, which was $\mathbf{V}^0$ in our argument, must be significantly weakened, however. If we could take $T^2_1(\alpha)$ as our base theory, we could reason about whether $Res(\log)$ has the feasible interpolation property in the same way as Krajíček and Pudlák [13], Bonet, Pitassi and Raz [4] or Bonet, Domingo, Gavaldà, Maciel, and Pitassi [3]. Now, if $Res(\log)$ does not have quasi-polynomial feasible interpolation we know by a result from Atserias and Bonet [2] that Resolution is not weakly automatizable, so we would be finished. Whether we can actually do it depends on the strength of the theory the polylogarithmic cut of $T^2_1(\alpha)$ models and if we can formalize some sort of iterated multiplication (such as in [11]) in that theory. Also, the security of Diffie-Hellman seems to be a more appropriate assumption than that of RSA, as the computational power needed to verify the correctness of Diffie-Hellman seems to be lower.

6. Acknowledgements

I want to thank Steve Cook, Jan Krajíček and Neil Thapen for helpful suggestions and discussion, Emil Jeřábek for his comments and for answering my questions and the participants of the MALOA Special Semester in Proof Complexity in Prague 2011 for enduring a sloppy and sometimes faulty exposition of this proof and still coming up with helpful comments. I also want to thank the anonymous referees for pointing out various mistakes and for giving interesting suggestions. A similar construction can be extracted from [20] and leads to similar results, if perceived in the way we did it here. I want to thank Leszek Kolodziejczyk for pointing this out.

References

[1] S. Arora and B. Barak. Computational Complexity. *Cambridge University Press*, 2009.
[2] A. Atserias and M. L. Bonet. On the Automatizability of Resolution and Related Propositional Proof Systems. *Information and Computation*, Vol. **189**(2), 2004, pp. 182-201.
[3] M.L. Bonet, C. Domingo, R. Gavaldà, A. Maciel, and T. Pitassi. Non-Automatizability of Bounded-Depth Frege Proofs. *Computational Complexity*, Vol. **13**, 2004, pp.47-68.
[4] M.L. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege systems. *SIAM Journal of Computing*, Vol. **29 (6)**, 2000, pp. 1939-1967.
[5] S. Buss. Polynomial Size Proofs of the Propositional Pigeonhole Principle. *Journal of Symbolic Logic*, Vol. **52**, 1987, pp. 916-927.
[6] S. Cook, K. Ghasemloo and P. Nguyen. Subexponential Size Bounded Depth Frege Proofs and Subsystems of TV^0. Manuscript, 2012.
[7] S. Cook and P. Nguyen. Logical Foundations of Proof Complexity. *Cambridge University Press*, 2010.
[8] S. Cook and R. Reckhow. The Relative Efficiency of Propositional Proof Systems. *Journal of Symbolic Logic*, Vol. **44**(1), 1979, pp.36-50.

[9] V. Chvátal and E. Szemerédi. Many Hard Examples for Resolution. *Journal of the Association for Computing Machinery*, Vol. **35**(4), 1988, pp. 759-768.
[10] Y. Filmus, T. Pitassi, and R. Santhanam. Exponential Lower Bounds for AC-Frege Imply Superpolynomial Frege Lower Bounds. *Proceedings ICALP*, Vol. **1**, 2011, pp.618-629.
[11] W. Hesse, E. Allender and D. Barrington. Uniform Constant-Depth Threshold Circuits for Division and Iterated Multiplication. *Journal of Computer and System Sciences*, Vol. **65**, 2002, pp.695-716.
[12] J. Krajíček. Bounded Arithmetic, Propositional Logic, and Complexity Theory. *Cambridge University Press*, 1995.
[13] J. Krajíček and P. Pudlák. Some Consequences of Cryptographical Conjectures for S_2^1 and EF. *Information and Computation*, Vol. **140 (1)**, 1998, pp.82-94.
[14] J. Krajíček, P. Pudlák and A. Woods. Exponential lower bound to the size of bounded depth Frege proofs of the Pigeon Hole Principle. *Random Structures and Algorithms*, Vol. **7**(1), 1995, pp.15-39.
[15] S. Müller and I. Tzameret. Short Propositional Refutations for Dense Random 3CNF Formulas. Manuscript, 2011.
[16] V.A. Nepomnjascij. Rudimentary Predicates and Turing Calculations. *Doklady AN SSSR*, Vol. **195**, 1970.
[17] J. Paris and A. Wilkie. Counting Problems in Bounded Arithmetic. *Methods in Mathematical Logic*, LNM **1130**, 1985, pp.317-340.
[18] J. Paris and A. Wilkie. Counting Δ_0 Sets. *Fundamenta Mathematica*, Vol. **127**, 1987, pp.67-76.
[19] T. Pitassi, P. Beame, R. Impagliazzo. Exponential Lower Bounds for the Pigeonhole Principle. *Computational Complexity*, Vol. **3**, 1993, pp.97-140.
[20] D. Zambella. End extensions of models of linearly bounded arithmetic. *Annals of Pure and Applied Logic*, Vol. **88**, 1997, pp.263-277.