Theories for Subexponential-size Bounded-depth Frege Proofs*

Kaveh Ghasemloo and Stephen A. Cook

Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada
{[first name],sacook}@cs.toronto.edu

Abstract

This paper is a contribution to our understanding of the relationship between uniform and nonuniform proof complexity. The latter studies the lengths of proofs in various propositional proof systems such as Frege and bounded-depth Frege systems, and the former studies the strength of the corresponding logical theories such as VNC^1 and V^0 in [7]. A superpolynomial lower bound on the length of proofs in a propositional proof system for a family of tautologies expressing a result like the pigeonhole principle implies that the result is not provable in the theory associated with the propositional proof system.

We define a new class of bounded arithmetic theories n^{ε}-ioV^{∞} for $\varepsilon < 1$ and show that they correspond to complexity classes $\mathsf{AltTime}(O(1), O(n^{\varepsilon}))$, uniform classes of subexponential-size bounded-depth circuits $\mathsf{DepthSize}(O(1), 2^{O(n^{\varepsilon})})$. To accomplish this we introduce the novel idea of using types to control the amount of composition in our bounded arithmetic theories. This allows our theories to capture complexity classes that have weaker closure properties and are not closed under composition. We show that the proofs of Σ_0^B-theorems in our theories translate to subexponential-size bounded-depth Frege proofs.

We use these theories to formalize the complexity theory result that problems in uniform NC^1 circuits can be computed by uniform subexponential bounded-depth circuits in [1]. We prove that our theories contain a variation of the theory VNC^1 for the complexity class NC^1. We formalize Buss's proof in [4] that the (unbalanced) Boolean Formula Evaluation problem is in NC^1 and use it to prove the soundness of Frege systems. As a corollary, we obtain an alternative proof of [10] that polynomial-size Frege proofs can be simulated by subexponential-size bounded-depth Frege proofs.

Our results can be extended to theories corresponding to other nice complexity classes inside $\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)})$ such as NL. This is achieved by essentially formalizing the containment

$$\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)}) \subseteq \mathsf{AltTime}(O(1), O(n^{\varepsilon}))$$

for all $\varepsilon > 0$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems - Complexity of proof procedures, F.4.1 Mathematical Logic – Proof theory

Keywords and phrases Computational Complexity Theory, Proof Complexity, Bounded Arithmetic, NC^1-Frege, AC^0-Frege

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.296

* Supported by NSERC. The research leading to these results has received funding from the European Union's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381.

1 Introduction

In [7] a general method is presented for associating a theory VC of bounded arithmetic and a (quantified) propositional proof system C-Frege with a complexity subclass C of the functions computable in polynomial time FP. The base complexity class is AC^0 of functions computable by a uniform family of polynomial-size bounded-depth Boolean circuits. The associated theory is V^0 and the associated proof system is bounded-depth Frege (bdFrege)[1]. Another important example is the complexity class NC^1 of functions computable by a uniform family of polynomial-size $O(\lg n)$-depth circuits (equivalently polynomial-size Boolean formulas), where the theory is VNC^1 and a Frege system serves as the associated proof system[2].

In general VC is obtained by adding a comprehension axiom for a function complete for C to the base theory V^0 [7]. Similarly, the proof system C-Frege is obtained by adding the cut rule for formulas in nonuniform C to the base proof system bdFrege [7, 11]. Moreover there are witnessing theorems stating that certain proofs in VC can be witnessed using C functions, and proofs in polynomial-size quantified C-Frege can be witnessed by nonuniform C functions. The witnessing theorems can be used to relate the functions in C and nonuniform C to the definable functions in VC and C-Frege [7, 8].

There is a two-fold connection between the theory VC and the propositional proof system C-Frege:

1. VC proves the soundness of C-Frege.
2. Any Σ_0^B theorem (essentially a universal theorem) of VC, translates into a polynomial-size family of tautologies with polynomial-size C-Frege proofs. For example the pigeonhole principle is provable in VNC^1, therefore its propositional translation has polynomial-size Frege proofs.

The second connection provides a universality condition that complements the first connection: C-Frege is maximal among propositional proof systems whose soundness can be proven in VC. More formally, C-Frege can efficiently prove any tautology family which is efficiently provable in any propositional proof system whose soundness is provable in VC [16, 6, 7]. Here a tautology family is efficiently provable in a proof system iff it has a polynomial-size proof family in the proof system, and polynomial size means polynomial size in the length of formulas being proven.

With this perspective, we consider the Σ_0^B fragment of the theory VC as a uniform version of the associated propositional proof system C-Frege.

One motivation for the present paper is the [FPS'11] result that Frege proofs can be simulated by subexponential-size bdFrege proofs. We wish to generalize this result and prove it in a uniform setting. The first step is to find a theory whose provably total functions are those computable by uniform families of bounded-depth subexponential-size circuits.

But here we run into a fundamental obstacle. A function f is subexponential[3] iff $f(n) = 2^{O(n^\varepsilon)}$ for some $\varepsilon < 1$. But the class of subexponential functions is not closed under composition, and not closed even under composition with polynomials. For example, if we compose the subexponential function $n \mapsto 2^{O(n^{\frac{1}{2}})}$ with the polynomial function $n \mapsto n^2$ the resulting function is $n \mapsto 2^n$, which is not subexponential. On the other hand, the provably

[1] bdFrege is also referred to as AC^0-Frege.
[2] Frege is also referred to as NC^1-Frege.
[3] There are other definitions of subexponential functions in the literature. The definition given here is the largest class among them. Using this version is required since even in computational complexity theory it is not known if the bounded-depth circuit classes of smaller size contain NC^1.

total functions in theories which extend the base theory V^0 are closed under composition[4].

To deal with this issue, we introduce two types of variables: input type and output type. The idea is that for a fast growing function f, the arguments of f have input types and are small, while the value of f has output type and might be large. This allows us to control the compositions of the provably total functions of theories. We refer to these typed theories as *io-typed* theories. We can then explicitly allow a limited amount of composition if we want. For example, since subexponential functions are closed under composition with linear functions, we can allow this limited amount of composition in a theory corresponding to subexponential size circuits by conversion axioms defined below in section 2.3.

The propositional proof translations from V^0 to $\mathsf{bdFrege}$ in [7] can be adapted for translating proofs from its io-typed version ioV^0 to $\mathsf{bdFrege}$. In addition, it is possible to prove the soundness of these proof systems in ioV^0: we have enough comprehension to define the truth of a sequent in a propositional proof, and we have enough induction to show that sequents in a proof are true under an arbitrary assignment.

Next, we define an extension $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ of ioV^0 (for any $\varepsilon < 1$ of the form $1/d$) whose provably total functions are those computed by $\mathsf{AltTime}(O(1), O(n^\varepsilon))$, a uniform succinct subclass of size $2^{O(n^\varepsilon)}$ bounded-depth circuits. We provide a translation from $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ to polynomial size proofs in the quantified proof system $n^\varepsilon\text{-}\mathsf{bdG}_\infty$, and also to subexponential-size $\mathsf{bdFrege}$ proofs. In addition, $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ can prove the soundness of $n^\varepsilon\text{-}\mathsf{bdG}_\infty$ proofs.

We observe that in general, for proving the soundness of line-based propositional proof systems like $\mathsf{C}\text{-}\mathsf{Frege}$ we use the following two ingredients:

- Evaluate the sequents in a proof, and state their truth under a given truth assignment.
- Use induction to prove the sequents in proofs are true under an arbitrary truth assignment.

The comprehension and induction axioms in our theories provide these ingredients.

The central role of soundness tautologies for proof complexity is similar to the central role of complete problems for computational complexity theory. The complete problems for complexity classes provide universal problems that can be used to solve any problem in those classes (over a class of reductions). A similar universal role is played by soundness tautologies in proof complexity: the soundness tautologies for that propositional proof system can be made to serve as axioms for the proof system.

Finally, we show that $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ contains ioVNC^1 and ioVNL, the io-typed versions of VNC^1 and VNL. It follows that $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ proves the soundness of Frege, and this gives us a uniform version of result in [10] that we can simulate Frege proofs with subexponential-size $\mathsf{bdFrege}$ proofs. The inclusion $\mathsf{ioVNC}^1 \subseteq n^\varepsilon\text{-}\mathsf{ioV}^\infty$ (and similar results) is proven essentially by formalizing inside $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ the following theorem[5] from computational complexity theory (for all $\varepsilon < 1$):

$$\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)}) \subseteq \mathsf{AltTime}(O(1), O(n^\varepsilon)).$$

The rest of the paper is organized as follows:

Section 2 We define our io-typed theories. We start with the base theory ioV^0, the io-typed version of V^0. We then extend ioV^0 to ioVNC^1, the io-typed version of VNC^1, which corresponds to the complexity class NC^1. Our theories $n^\varepsilon\text{-}\mathsf{ioV}^\infty$ corresponding to (uniform) subexponential-size bounded-depth circuits are also defined in this section.

Section 3 We provide the background information about propositional proof systems. We introduce proof classes and define the proof class polynomial-size $n^\varepsilon\text{-}\mathsf{bdG}_\infty$.

[4] This obstacle also arises in attempts to design a theory that corresponds to small classes like fixed-depth circuits and propositional proof systems like $\mathsf{Resolution}$.

[5] The result is a similar to Nepomnjaščij's theorem [15, 5].

Section 4 We provide a propositional translation from n^ε-ioV^∞ to n^ε-bdG_∞ and subexponential-size $\mathsf{bdFrege}$.

Section 5 We discuss the soundness of $\mathsf{bdFrege}$, Frege, and n^ε-bdG_∞ in io-typed theories. We discuss the provability of Buss's result [4] that Boolean Formula Evaluation can be computed in NC^1 in our theory ioVNC^1 and use it to show that ioVNC^1 can prove the soundness of Frege.

Section 6 We show that n^ε-ioV^∞ contains ioVNC^1. Theorem 28 is our proof complexity version of the computational complexity theory result that the uniform NC^1 can be computed by $\mathsf{AltTime}(O(1), O(n^\varepsilon))$, a uniform complexity class corresponding to subexponential-size bounded-depth circuits [1]. It is also a uniform version of [10] that polynomial-size Frege proofs can be simulated by subexponential-size $\mathsf{bdFrege}$ proofs.

	Nonuniform	Uniform
Computational Complexity	$\mathsf{NC}^1/\mathsf{poly} \subseteq \mathsf{DepthSize}(O(1/\varepsilon), 2^{O(n^\varepsilon)})$ (Folklore)	$\mathsf{NC}^1 \subseteq \mathsf{AltTime}(O(1), O(n^\varepsilon))$ [1], (Omitted Appendix)
Proof Complexity	$\mathsf{FregeSize}(n^{O(1)}) \subseteq O(1/\varepsilon)$-$\mathsf{FregeSize}(2^{O(n^\varepsilon)})$ [10]	$\mathsf{ioVNC}^1 \subseteq n^\varepsilon$-$\mathsf{ioV}^\infty$ (Theorem 28)

Section 7 We combine the results in the previous sections to show that polynomial-size Frege proofs are simulated by subexponential-size $\mathsf{bdFrege}$ proofs and to provide an alternative proof of [10].[6]

2 A Theory for Subexponential-Size Bounded-Depth Circuits

We start by defining an io-typed version of LK and the base theory $\mathsf{2Basic}$ for two-sorted bounded arithmetic in [7].

2.1 Two-Sorted io-Typed Bounded Arithmetics

Our language $\mathcal{L}_2$ have two sorts: num for (unary) numbers, and str for (binary) strings[7]. In addition, we will syntactically type the terms of the language as[8]: i for input type, and o for output type. The input types are subtypes of the output types, i.e. every object of input type is also an object of output type: $\mathsf{num}^\mathsf{i} \subseteq \mathsf{num}^\mathsf{o} = \mathsf{num}$, $\mathsf{str}^\mathsf{i} \subseteq \mathsf{str}^\mathsf{o} = \mathsf{str}$.

We refer to the free variables of a formula as its *parameters*. The idea here is that the input-type terms are going to be *small* (of linear size in the parameters) while the output-type terms can be *large* (of polynomial size in the parameters, like the original two-sorted theories of bounded arithmetic).

Lower-case letters denote numbers: a, b, c denote input-type numbers, and x, y, z denote output-type numbers. Upper-case letters denote strings: A, B, C denote input-type strings, and X, Y, Z denote output-type strings. We use f, g for number-valued functions; D, F, G for string-valued functions; s, t for number-values terms; T for string-valued terms; φ, ψ, etc. for formulas; $\Sigma, \Gamma, \Delta, \Pi$ for set of formulas; S for sequents; and T for sets of sequents, i.e. theories.

[6] A similar result is obtained independently in [14] using model theoretic methods.

[7] Or equivalently, finite sets of natural numbers with explicit (strict) upper bounds on the their members. For example, the string 01100 is equivalent to the finite set $\{2, 3\}$ with the explicit upper bound 5.

[8] We can view our theory as having a two-sorted theory with two types. Each object has a sort and a type. The sorts are: (unary) numbers and (binary) strings. The types are: input and output. Finally, the input type is a subtype of the output type. Alternatively, we can define it as a four-sorted language with appropriate axioms expressing the relation between them.

The language $\mathcal{L}_2$ has function symbols 0, 1, $+$ (addition), $\cdot$ (multiplication), pd (predecessor), $|\ |$ (length), and relation symbols $=$ (equality), $\leq$ (comparison), and $\in$ (membership/bit). All of these symbols except $\in$ and $|\ |$ apply to terms of number sort and have the obvious intended interpretations. The membership relation $y \in X$ means that the yth bit of the string X is one[9]. The length function $|X|$ returns the length of the string X as a number. We consider the size of a number to be the number itself. The size of a string is its length.

An $\mathcal{L}_2$-stricture has four sets for interpreting num, $\mathsf{num}^{\mathsf{i}}$, str, and $\mathsf{str}^{\mathsf{i}}$, where $\mathsf{num}^{\mathsf{i}} \subseteq \mathsf{num}$ and $\mathsf{str}^{\mathsf{i}} \subseteq \mathsf{str}$. The standard model $\mathbb{N}_2$ for $\mathcal{L}_2$ is given by interpreting num and $\mathsf{num}^{\mathsf{i}}$ as 0^*10^* and str and $\mathsf{str}^{\mathsf{i}}$ as $\{0,1\}^*$. Note that we encode unary numbers using binary strings with a single 1 bit whose location index from right determines the number [10]. We use m, and n for numbers and M and N for strings in the standard model. The operations and relations of $\mathcal{L}_2$ has the usual interpretations as explained above. Note that $m \in N$ is 0 for $|N| \leq m$.

A *linear term* is a term built from constants 0 and 1, variables, and $+$. A function has *provably linear growth* if the size of its output is provably bounded with a linear term in the size of its inputs.

Every term in the language is an output-type term. The input-type terms of the language are a subset of output-type terms and have provably linear growth. Input-type variables are input-type terms, and when functions 0, 1, $|\ |$, $+$, pd are applied to input-type terms, the result is also of input-type. Formally, the input-type terms are defined inductively: input-type terms include input-type variables and constants 0 and 1; input-type terms are closed under $|\ |$, $+$, and pd.

Unless stated otherwise, by quantifiers we mean quantifiers of both types. In one-sorted theories, the bounded formulas with at most i alternations of number quantifiers comprise the union of the classes Σ^b_i and Π^b_i. In two-sorted theories, these classes are defined similarly and do not have any string quantifiers, but can have free string variables. We will often be interested in the class of number bounded formulas Σ^B_0. A formula is Σ^B_0 if it does not have any string quantifiers and all number quantifiers in it are bounded by terms in the language. We abbreviate $|X| = y$ and $|X| \leq y$ by $X = y$ and $X \leq y$. A bounded string quantifier is a string quantifier with an explicitly given size e.g. $\exists X = y$. Bounded formulas with at most i alternations of string quantifiers comprise the union of the classes Σ^B_i and Π^B_i. We define $\Sigma^B_\infty = \bigcup_i \Sigma^B_i$. Let Φ be a class of formulas. The formula class $\exists^B\Phi$ consists of formulas starting with bounded existential string quantifiers followed by a formula in Φ. We will be using a subclass of bounded formulas which we call $\Sigma^B_\infty(t(n))$.

► **Definition 1** ($\Sigma^B_\infty(t(n))$). We call a formula $\Sigma^B_\infty(t(\vec{n}))$ iff it is Σ^B_∞ and all of its string quantifiers are bounded by number terms of size $O(t(\vec{n}))$ where $\vec{n}$ is the size of its free variables. We often refer to this class simply as $\Sigma^B_\infty(t(n))$ in place of $\Sigma^B_\infty(t(\vec{n}))$. In such cases n can be considered to be the total size of free variables.

Note that every formula Σ^B_0 represents a relation $R(\vec{x}, \vec{X})$ in its free variables in the standard model. These relations comprise the complexity class $\mathsf{AC}^0 = \mathsf{AltTime}(O(1), O(\lg n))$ [7].

[9] Our semantics differs slightly from [7] where the second sort objects are finite subsets of $\mathbb{N}$, and technically are binary strings starting with 1. Our second sort objects are finite binary strings, where the most significant bit does not need to be 1.

[10] This more complex encoding is needed since otherwise we cannot consider nonconstant number-valued functions in nonuniform models of computation. We need to be able to represent different unary number with the same number of bits. The reason for choosing this particular encoding is its efficiency for performing operations like checking the value of a given unary number. We discard the leading zeros when considering these strings as numbers. For example, 00010 and 010 both represent number 1 and are equal.

We adopt the sequent calculus LK of [7] with quantifier introduction rules to respect the types. In the quantifier introduction rules for input types the target term must be an input-type term. Similarly, if the quantifier variable is of output type, the eigenvariable must be an output-type variable. The intuition here is that if we are deriving the existence of a small object with some property, we must have a small object satisfying the property; or if we are deriving that a property holds for all objects, the property must hold for an arbitrary object, not just small ones.

More formally, in $\exists\mathsf{R}$ and $\forall\mathsf{L}$ rules, if the quantification variable is of input type then the target term must be also of input type. Similarly, in $\forall\mathsf{R}$ and $\exists\mathsf{L}$ rules, if the quantification variable is of output type, then the eigenvariable must be also of output type. These restrictions make sure that we *cannot* derive $\forall a\ \varphi(a) \Rightarrow \forall x\ \varphi(x)$ and $\exists x\ \varphi(x) \Rightarrow \exists a\ \varphi(a)$. The implications in the other direction are still provable as expected: input types are subsets of output types, so $\forall x\ \varphi(x) \Rightarrow \forall a\ \varphi(a)$ and $\exists a\ \varphi(a) \Rightarrow \exists x\ \varphi(x)$ are valid.

We write $\pi : \mathrm{T} \vdash \varphi$ for "π is a LK-proof of φ in the theory T", and $\mathrm{T} \vdash \varphi$ for "φ has an LK-proof in the theory T". We refer to the free variables of the end-sequent of an LK proof as its *parameters*.

Let Φ be a set of formulas (for example take $\Phi = \exists^B\Sigma_0^B$). We say that a set is Φ-definable (over the standard model) iff there is a formula $\varphi \in \Phi$ which defines the set over the standard model. The graph of a function f is defined as $\{(\vec{n}, \vec{N}, m) \in \mathbb{N}_2 \mid f(\vec{n}, \vec{N}) = m\}$. We say that a function f is Φ-definable in T iff its graph is Φ-definable using a formula $\varphi(\vec{x}, \vec{X}, z) \in \Phi$ and T proves that φ defines a function, i.e. $\mathrm{T} \vdash \exists! y\ \varphi(\vec{x}, \vec{X}, y)$.

Let F be string-valued. The bit-graph of F is defined as $\{(\vec{n}, \vec{N}, m) \in \mathbb{N}_2 \mid m \in F(\vec{n}, \vec{N})\}$. We say a string-valued function F is Φ-bit-definable in a theory T iff there is a formula $\varphi(\vec{x}, \vec{X}, z) \in \Phi$ and a number term $p(\vec{x}, \vec{y})$ such that[11]

- $|F(\vec{n}, \vec{N})| = p(\vec{n}, |\vec{N}|)$,
- φ defines the bit-graph of the function F (over the standard model), and
- T proves that φ and p define a total function, i.e.

$$\mathrm{T} \vdash \exists! Y = p(\vec{x}, |\vec{X}|)\ \forall z < p(\vec{x}, |\vec{X}|)\ \Big(z \in Y \leftrightarrow \varphi(\vec{x}, \vec{X}, z)\Big)\,.$$

Notice that the size of F depends only on the size of its arguments. We say that a function is *provably total* in a theory T if the function is Φ-definable in T, for the appropriate choice of the function class Φ. The choice of Φ depends on the theory T: We want the provably total functions in T to be those in the complexity class associated with T. For two-sorted theories associated with complexity classes contained in polynomial time (such as ioV^0 defined below), the right choice is $\Phi = \exists^B\Sigma_0^B$ [7]. For the theory $t(n)$-ioV^∞ defined in Section 2.4 we choose Φ to be a larger class.

When a function is provably total in a theory T, we can add a new function symbol to the language for it and include its definition as an axiom to our theory to obtain a conservative extension of T. This extends the language and possible formulas. We are interested in whether axiom schemas like comprehension continue to hold for the extended class of formulas. For cases of interest in this paper the techniques presented in [7] suffice to show this.

When we extend the language by adding a new provably total function symbol, if we can prove that the function has linear growth, then we can extend input type terms to be closed under the new function symbol.

[11] Note that in circuit complexity, the size of output of a function must only depend on the size of its inputs.

Table 1 io2Basic.

B1	$x + 1 \neq 0$	B7	$x \leq y \wedge y \leq x \to x = y$
B2	$x + 1 = y + 1 \to x = y$	B8	$x \leq x + y$
B3	$x + 0 = x$	B9	$0 \leq x$
B4	$x + (y + 1) = (x + y) + 1$	B10	$x \leq y \vee y \leq x$
B5	$x \cdot 0 = 0$	B11	$x \leq y \leftrightarrow x < y + 1$
B6	$x \cdot (y + 1) = x \cdot y + x$	B12	$\mathsf{pd}(0) = 0 \wedge (x \neq 0 \to \mathsf{pd}(x) + 1 = x)$
L	$y \in X \to y < \lvert X \rvert$		

The class of io-typed provably total functions of a theory T are those functions that T can prove to be total when the inputs to the functions are of input type. More formally, the free variables in φ corresponding to inputs have input type. In other words, it is sufficient for the function to be total over input-type objects. This is the class of functions we associate with a theory. Note that this class is a possibly larger class than the usual class of provably total functions of a theory since every input-type object is also an output-type object.

Equality for strings, $X = Y$, is an abbreviation for $|X| = |Y| \wedge \forall x \leq |X| \;\; x \in X \leftrightarrow x \in Y$. The notations $X[y, z]$ and $X[i; l]$ abbreviate the substring of X starting from bit y up of length z, and ith substring block of X of length l:

$$x \in X[y, z] := x < z \wedge y + x \in X, \qquad |X[y, z]| := z, \qquad X[i; l] := X[i \cdot l, l].$$

2.2 Theory io2Basic

The axioms of io2Basic are given in table 1.

Note that unlike the original 2Basic in [7], our length function $|\;|$ gives only an upper bound on the size of binary numbers. In this sense our axioms are similar to the second-order theories in [2]. A binary string is determined by its length and its bits. This change doesn't make any essential difference in the presence of number induction for Σ_0^B formulas: the original version of the length function is definable.

Let d be a fixed positive integer. The *fractional power* $\lfloor x^{\frac{1}{d}} \rfloor$, which we will write simply as $x^{\frac{1}{d}}$, is definable using $x^{\frac{1}{d}} = y \leftrightarrow y^d \leq x < (y + 1)^d$.

2.3 Theory ioV^0 for AC^0

Our theory ioV^0 is an io-typed version of the base theory V^0 of [7]. Besides the axioms for io2Basic, we need an io-typed axiom for induction, an io-typed axiom scheme for comprehension, and two conversion axioms (one for each sort). *All* free variables in the axioms below which are not displayed are of input type[12].

- Ind: $0 \in X, \forall y < z \;\; (y \in X \to y + 1 \in X) \Rightarrow z \in X$
- φ-CA: $\Rightarrow \exists Y = z \;\; \forall x < z \;\; (x \in Y \leftrightarrow \varphi(x, A))$

Although we do not want to allow arbitrary compositions, we may want to allow some under specific conditions. The main condition of interest for us here is to avoid increasing the size of the input strings. Therefore, we will add conversion axioms that would allow us to create composition when the size of the computed intermediate values are small[13].

[12] We could have used a stronger version of Σ_0^B-CA where the free variables have output type. In that case the input-type variable free part of the theory will be the same as V^0. The simpler axiom is sufficient and allows a conceptually and technically cleaner treatment.

[13] There are other ways of expressing this axiom e.g. a comprehension axiom from output types to input types, however these variations do not affect our results and we take this simpler from.

- $\mathsf{oiConv_{num}}$: $\Rightarrow \exists b \leq a \ \ (b = x \leq a) \vee (b = a \leq x)$
- $\mathsf{oiConv_{str}}$: $\Rightarrow \exists B = a \ \forall z < a \ \ (z \in B \leftrightarrow y + z \in X)$

We refer to these two axioms together as oiConv. The first axiom tells us that the minimum of two numbers is small when at least one of them is small. The second axiom tells us that a small substring of an output type string is small. These axioms allow us to compose definable functions if the intermediate results are small in some input variables.

The theory ioV^0 is obtained from io2Basic by adding the comprehension axiom for Σ_0^B formulas, the induction axiom, and the conversion axioms.

▶ **Definition 2.** $\mathsf{ioV}^0 = \mathsf{io2Basic} + \mathsf{Ind} + \Sigma_0^B\text{-}\mathsf{CA} + \mathsf{oiConv}$.

As noted earlier the sets in $\mathsf{AltTime}(O(1), O(\lg n)) = \mathsf{LH} = \mathsf{FO} = \mathsf{AC}^0$ are precisely the sets definable by Σ_0^B formulas. Since ioV^0 has comprehension for these sets, p-bounded functions with bit graphs in these sets are $\exists^B\Sigma_0^B$ definable functions in the theory. By a witnessing theorem, they are the only $\exists^B\Sigma_0^B$ definable functions in the theory. Thus the provable total functions in ioV^0 coincide with the AC^0 functions, where we take the class Φ associated with this theory to be $\exists^B\Sigma_0^B$. As a general rule, the comprehension axiom of any of our theories will determine its computational power.

2.4 Theory ioVC

We can define io-typed versions of other theories built upon V^0. However, simply adding the same comprehension axiom used for VC in [7] might not be sufficient. The io-typed version of these theories can be weaker than their original version. The io-types do not allow arbitrary compositions of the provably total functions of a theory. Therefore, adding the comprehension axiom for a problem complete with respect to AC^0 reductions might not capture the complexity class. This is intentional and necessary since we are going to deal with complexity classes which are not closed under composition (they are not closed even under composition with AC^0 reductions from the right, e.g. consider the subexponential-size bounded-depth circuits where their AC^0 closure contains all functions via padding). Therefore, we cannot define an io-typed theory assuming that the provably total functions of the theory are closed under composition. For example, the theory VTC^0 captures TC^0 because every TC^0 function can be built by composing a finite number of MAJ[14] and AC^0 functions, $\mathsf{TC}^0 = \overline{\mathsf{MAJ} + \mathsf{AC}^0}$. This result is not useful for defining the io-typed version of the theory. Or the theory VNC^1 captures NC^1 because $\mathsf{NC}^1 = \mathsf{MBBFE} \circ \mathsf{AC}^0$ where MBBFE is the Monotone Balanced Boolean Formula Evaluation problem.

With this in mind we have to be careful about the representations of complexity classes we use in the comprehension axiom. The main requirement for a reasonable theory ioVC for computational complexity class C is that ioVC has enough comprehension to evaluate problems in C and the provably total functions of ioVC are exactly the functions in C.

2.5 Theory ioVNC^1

▶ **Definition 3.** The io-typed version of the theory VNC^1, ioVNC^1 is defined as $\mathsf{ioV} + \Sigma_0^B(\mathsf{MBBFE})\text{-}\mathsf{CA}$, where $\Sigma_0^B(\mathsf{MBBFE})\text{-}\mathsf{CA}$ is

$$\exists Y = 2s \ \exists Z = 2s \ \ [\forall x < 2s \ \ (x \in Z \leftrightarrow \varphi(x, A)) \wedge \text{“}Y \text{ is the computation of } Z\text{”}]$$

[14] The function MAJ computes the majority for the rows of a given matrix.

We think of Z as an instance of MBBFE and its second half gives inputs to the formula. "Y is the computation of Z" is a shorthand for

$$\forall z < s\ [(z \in Z \to (z \in Y \leftrightarrow 2z \in Y \wedge 2z+1 \in Y)) \wedge$$
$$(z \notin Z \to (z \in Y \leftrightarrow 2z \in Y \vee 2z+1 \in Y))]$$

The theory ioVNC^1 corresponds to NC^1. We will look at $\exists^B\Sigma_0^B$ theorems where the free variables have input type and existentially quantified string variables have output-type. For example, consider the identity function that maps an input string to an output string of the same value. This can be expressed by the formula $\exists X\ X = A$. This formula is provable in ioV^0 using the comprehension axiom. We have the following theorem:

▶ **Theorem 4.** *The $\exists^B\Sigma_0^B$ definable functions of ioVNC^1 are precisely NC^1 functions.*

Proof. The proof is similar to the proof for VNC^1 in [7]. For one direction we note that the witnessing theorem still applies.

For the other direction any NC^1 function is can be obtained by composing MBBFE with an AC^0 function. We can express this using a $\exists^B\Sigma_0^B$ formula: $\Sigma_0^B(\mathsf{MBBFE})$ without the leftmost quantifier and with a suitable φ defining the AC^0 function. By comprehension axiom for $\Sigma_0^B(\mathsf{MBBFE})$ the function is provably total. ◀

2.6 Theory $t(n)$-ioV^∞

Next, we define our theory for the complexity classes $\mathsf{AltTime}(O(1), O(t(n)))$. We are interested in $t(n) = n^\varepsilon$ where $\varepsilon = \frac{1}{d} < 1$ for some fixed d. Functions in $\mathsf{AltTime}(O(1), O(t(n)))$ can be computed by uniform families of subexponential-size bounded-depth circuits. Note that every function in $\mathsf{AltTime}(O(1), O(t(n)))$ is definable by a $\Sigma_\infty^B(t(n))$ formula, i.e. a formula with string quantifiers bounded by a term of size $O(t(n))$ where n is a bound on the size of inputs. The complexity class $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ corresponds to $\Sigma_\infty^B(n^\varepsilon)$ in a similar way that the complexity class $\mathsf{AltTime}(O(1), O(\lg n)) = \mathsf{AC}^0$ corresponds to $\Sigma_\infty^B(\lg n) = \Sigma_0^B$. The complexity class $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ is a nice uniform version of subexponential-size bounded-depth circuits. We will include the comprehension axiom $\Sigma_\infty^B(t(n))$-CA for this class of functions to provide the necessary computational power. Note that $t(n)$ is a term of number sort and input-type which bounds the quantified string variables in φ. The term $t(n)$ cannot contain any output-type variable. We will consider cases where $t(n)$ is not a term in the original language but an AC^0 function definable in the base theory ioV^0 with at most linear growth. In these cases we can easily extend the language to contain the new function and add the defining axiom of the function to the theory, e.g. $t(n) = n^{\frac{1}{d}}$ where $n = |A|$.

We define our theory $t(n)$-ioV^∞ as follows:

▶ **Definition 5.** $t(n)$-$\mathsf{ioV}^\infty = \mathsf{ioV}^0 + \Sigma_\infty^B(t(n))$-$\mathsf{CA}$

It is easy to see that ioV^0 is equivalent to $\lg n$-ioV^∞ since we can convert unary numbers to binary numbers of logarithmic size and vice versa in AC^0. Using Σ_0^B-CA and definability of *Bit* in ioV^0 we can prove $\Sigma_\infty^B(\lg n) = \Sigma_0^B$.

We take the provably total functions in $t(n)$-ioV^∞ to be the Φ-definable functions, where $\Phi = \exists^B\Sigma_\infty^B(t(n))$.

▶ **Theorem 6** (Provably Total Functions of $t(n)$-ioV^∞). *The provably total functions of the theory $t(n)$-ioV^∞ are exactly those in $\mathsf{AltTime}(O(1), O(t(n)))$ of size $n^{O(1)}$, where n is the size of the arguments.*

Proof Outline. The proof is similar to the proof for V^0: functions in $\mathsf{AltTime}(O(1), O(t(n)))$ are definable and provably total using the comprehension axiom for $\Sigma^B_\infty(t(n))$. On the other hand, by a witnessing theorem every provably total function of $t(n)$-ioV^∞ is in $\mathsf{AltTime}(O(1), O(t(n)))$. ◀

3 Proof Systems and n^ε-bdG_∞ Proofs

We use lower-case letters like p and q for propositional variables; and α, β, γ for propositional function symbols. We use φ, ψ, etc. for formulas (propositional, quantified propositional, and first-order).

We say that a term is free in a formula iff all of its variables are free in the formula. Expressions like $\varphi[p/q]$ denote the formula resulting from substituting q for p in φ. The usual restrictions on substitution apply, e.g. only free occurrences are replaced, and new variables must not become bound after substitution.

We allow unbounded $\bigwedge$ and $\bigvee$ connectives in propositional formulas. The (logical) depth of a propositional formula denoted by $\mathsf{ldepth}()$ is defined as the depth of the formula tree. The size of a propositional formula denoted by $\mathsf{size}()$ is the number of nodes in its tree[15]. Quantified propositional formulas are defined by allowing quantification over propositional variables of propositional formulas. We allow a single quantifier to quantify over multiple propositional variables. For quantified propositional formulas the depth is defined as the depth of their quantifier-free part. The quantifier depth of a quantified propositional formula is defined as the maximum depth of formula tree counting only quantifier nodes and is denoted by $\mathsf{qdepth}()$. The size of a proof is the total size of its formulas. The depth of a proof is the maximum depth of its formulas. The quantifier depth of a proof is the maximum quantifier depth of its formulas. The class of quantified propositional formulas with quantifier depth i starting with an existential quantifier is denoted by Σ^q_i. Σ^q_∞ is their union.

3.1 Proof Systems and Proof Classes

Let L denote a class of formulas, e.g. propositional formulas or quantified propositional formulas. Let τ be a truth assignment for free variables, i.e. a function from free variables to $\{0, 1\}$. We sometimes call τ an evaluation context. We use $\tau \vDash \varphi$ to express that a formula $\varphi \in L$ is true under the truth assignment τ. When φ is true under all truth assignments we write $\vDash \varphi$ and say that φ is valid. We refer to the set of valid formulas in L as L-tautologies and denoted it by $TAUT_L$.

Let Q be a relation with two inputs. We say π is a Q-proof for φ iff $\mathrm{Q}(\pi, \varphi)$ accepts, in which case we write $\pi : \mathrm{Q} \vdash \varphi$. We say φ is provable in Q iff φ has a Q-proof. An relation Q is a(n) (efficient) proof system for L iff

- efficiency: Q is computable in polynomial time,
- completeness: every L-tautology is provable in Q,
- soundness: every L-formula provable in Q is an L-tautology.

In propositional proof complexity we want to study *families* of proofs for *families* of formulas. We say a proof family $\{\pi_n\}_n$ is a Q-proof for a formula family $\{\varphi_n\}_n$ iff for all n, $\pi_n : \mathrm{Q} \vdash \varphi_n$. We define *proof classes* in a similar way to nonuniform computational complexity classes. Let Q be a proof system. A Q-proof class is a set of Q-proof families.

[15] If we encode formulas as binary strings there will be a constant factor of the number of nodes in the tree.

Similar to bounded-depth circuits, bounded-depth Frege (bdFrege) proof class is defined as the set of Frege-proof families where the depth of cut formulas is $O(1)$. Polynomial-size bdFrege is the subclass of bdFrege where the size of the proofs are polynomial in the size of the proven formula. If F is a proof class, we write $\mathsf{F} \vdash \{\varphi_n\}_n$ to state that $\{\varphi_n\}_n$ has a F-proof. For example, the pigeon-hole principle $PHP = \{PHP_n\}_n$ has polynomial-size Frege proofs [7, 12] but it does not have polynomial-size bdFrege proofs [12].

The proof class bdFrege is sometimes defined as the union of proof systems d-Frege for $d \in \mathbb{N}$, where d-Frege is a subsystem of Frege obtained by restricting cuts to depth d formulas. It is easy to see that this gives the same bdFrege proof class we defined above. Note that bdFrege is not a proof system, The proof system obtained from taking the union of proof systems d-Frege for $d \in \mathbb{N}$ is not bdFrege but Frege. In fact, it doesn't make much sense to say a single proof has a bounded depth.

3.2 Standard Proof Systems and Proof Classes: bdFrege, Frege, and G

Our reference is [7]. PK is the classical sequent calculus propositional proof system. For $d \in \mathbb{N}$, the propositional proof system d-PK is PK with cuts restricted to depth d formulas. bdPK is the proof class resulting from taking the union of d-PKs, i.e. proof families with the depth $O(1)$ cut formulas. Frege denotes any Frege proof system, e.g. PK. Bounded-depth Frege denoted by bdFrege (a.k.a. AC^0-Frege) is the Frege where cuts are restricted to bounded-depth formulas.

Next, we define proof systems for quantified propositional calculus. The proof system G is obtained from PK by adding the Boolean quantifier introduction rules. In the rules $\forall$R and $\exists$L, p is a free variable called *eigenvariable* and does not appear in the bottom sequent. In the rules $\forall$L and $\exists$R, $\varphi[q/\psi]$ is the result of substituting ψ for q in φ. The formula ψ is called the *target* formula of the rule and may be any quantifier-free formula[16]. The formulas $\exists q \ \varphi$ and $\forall q \ \varphi$ are called the *principal* formulas and the corresponding $\varphi[q/\psi]$ or $\varphi[q/p]$ formulas on top are called the *auxiliary* formulas. $\mathsf{G_i}$ is a subsystem of G where the cuts are restricted to formulas of quantifier depth i. For $d \in \mathbb{N}$, the proof class d-$\mathsf{G_i}$ is $\mathsf{G_i}$ where the depth of the quantifier free part of cut formulas is bounded by d. $\mathsf{bdG_i}$ is the union of these proof systems, i.e. the depth of the cut formulas in the proofs are bounded by a constant. $\mathsf{G_0}$0 is a conservative extension of Frege and $\mathsf{bdG_0}$ is a conservative extension of bdFrege [7, 12].

In our systems, we allow the introduction of a quantifier over multiple propositional variables in a single step. For example, we can derive $\exists \vec{p} \ \varphi(\vec{p})$ is a single step from $\varphi(\vec{\psi})$. Similarly, we allow the introduction of conjunction/disjunction of multiple formulas in a single step. These modifications do not change the power of the proof systems, but will be convenient to assume to obtain a nicer correspondence with first-order proofs.

In general, a proof can be a DAG and does not need to be a tree. We use a superscript * to denote proofs which are trees [7, p. 195].

3.3 Proof Systems n^ε-bdG_∞ and H

Let $\mathsf{bd}\Sigma^q_\infty(t(n))$ denote the class of those Σ^q_∞ formula families where the number of quantified propositional variables is bounded by $O(t(n))$ and the depth of quantifier-free part is bounded.

[16] The exact class of formulas for these rules is not important. By [7, VII.3.6, p.176], we can assume that the target formulas are arbitrary formulas and need not to be quantifier free. Similarly, we can restrict them to be only $\top$ or $\bot$. These modifications does not change the power of the proof system. Following [7], we use the definition of G (and its subsystems) that does not restrict all formulas but only the cut formulas.

▶ **Definition 7** ($t(n)$-bdG_∞)**.** The proof class $t(m)$-bdG_∞ is the class of bdG_∞ proofs for formula families where cuts are restricted to $\mathsf{bd}\Sigma^q_\infty(t(m))$ formulas where m is the size of proven formula. In addition we will assume that[17] the total number of eigenvariables in each sequent of in $t(m)$-bdG_∞ proofs must not exceed $t(m)$.

▶ **Definition 8** (H)**.** The proof system H is an extension of G obtained by allowing propositional function symbols (denoted by α, β, γ) as atomic formulas and including the following extensionality axiom for them[18] Ext: $\vec{p} \leftrightarrow \vec{q} \Rightarrow \alpha(\vec{p}) \leftrightarrow \alpha(\vec{q})$

We will use the propositional function symbols to remove quantifiers from the axioms and obtain Skolemized axioms. We will consider proofs with non-logical axioms.

4 From Uniform to Nonuniform

4.1 Translation of Terms and Formulas

In this section, we define a translation from first-order terms and bounded formulas to families of sequences of propositional formulas and quantified propositional formulas, respectively.

We use $(w)_i$ to refer to the ith item in the sequence w. For example, if $\vec{p} = (p_0, \dots, p_{k-1})$ is a sequence of propositional variables, then $(\vec{p})_i$ is p_i for $i < k$ (and is $\perp$ if $i \geq k$). Recall that s and t denote number terms and T denotes a string term.

Let Var be the set of variables. Let σ be a function that determines the size of variables, i.e. $\sigma : Var \to \mathbb{N}$ assigns a numeric value to each variable. We refer to σ as a translation context. The notation $\sigma[x \mapsto n]$ is used for the function obtained from σ by mapping x to n. The propositional translation of a term t and a formula φ under translation parameters σ are denoted by $[\![t]\!]_\sigma$ and $[\![\varphi]\!]_\sigma$. We sometimes use $[\![t]\!]_{\vec{n}}$ and $[\![\varphi]\!]_{\vec{n}}$ in place of $[\![t]\!]_{[\vec{x}\mapsto\vec{n}]}$ and $[\![\varphi]\!]_{[\vec{x}\mapsto\vec{n}]}$ when it is clear which variables $\vec{x}$ are being mapped.

The terms are translated recursively. The main difference from the usual propositional translations is that we have function symbols that we translate to propositional function symbols. We translate a function symbol into a sequence of propositional function symbols. Each of these propositional function symbols will correspond to a bit of the original function symbol. The number of propositional function symbols is the length of the original function symbol. We can translate functions of the theory that are AC^0 computable into reasonable AC^0 circuits computing them (sequences of AC^0 formulas).

We first need to extend the translation context σ to all terms in the language. We will use σ to determine the size of sequences used for the translation of the terms. The number of bits used for translating a term may only depend on the size of its variables. Note that every function symbol in our languages has an explicit size in terms of the size of its inputs. We denote the size of a function symbol by adding a superscript $^\sigma$. The translation context σ distributes over sequences, i.e. $\sigma(t_k, \dots, t_0) = \sigma(t_k), \dots, \sigma(t_0)$.

▶ **Definition 9** (Extended translation context)**.** Let σ be a translation context. The extended translation context is given in table 2. For any term t, $\sigma(t)$ is bounded by a polynomial in σ. If a variable x occurs in t, then $\sigma(t) \geq \sigma(x)$.

[17] This assumption simplifies some arguments significantly. However, we think that the results hold also without this assumption.
[18] A similar system is presented in [9].

Table 2 Extended Translation Context σ and Translation of Terms.

	$\llbracket x \rrbracket_\sigma = (\top, \overbrace{\bot, \dots, \bot}^{\sigma(x) \text{ times}})$
	$\llbracket X \rrbracket_\sigma = (p^X_{\sigma(X)-1}, \dots, p^X_0)$
$\sigma(0) = 0$	$\llbracket 0 \rrbracket_\sigma = (\top)$
$\sigma(1) = 1$	$\llbracket 1 \rrbracket_\sigma = (\top, \bot)$
$\sigma(t + s) = \sigma(t) + \sigma(s)$	$\llbracket s + t \rrbracket_\sigma = (o_{\sigma(s+t)}, \dots, o_0)$
	where $o_k = (\llbracket s + t \rrbracket_\sigma)_k = \bigvee_{\substack{i \leq \sigma(s), j \leq \sigma(t) \\ i+j=k}} (\llbracket s \rrbracket_\sigma)_i \wedge (\llbracket t \rrbracket_\sigma)_j$
$\sigma(t \cdot s) = \sigma(t) \cdot \sigma(s)$	$\llbracket s \cdot t \rrbracket_\sigma = (o_{\sigma(s \cdot t)}, \dots, o_0)$
	where $o_k = (\llbracket s \cdot t \rrbracket_\sigma)_k = \bigvee_{\substack{i \leq \sigma(s), j \leq \sigma(t) \\ i \cdot j=k}} (\llbracket s \rrbracket_\sigma)_i \wedge (\llbracket t \rrbracket_\sigma)_j$
$\sigma(\mathsf{pd}(t)) = \sigma(t)$	$\llbracket \mathsf{pd}(s) \rrbracket_\sigma = (o_{\sigma(\mathsf{pd}(t))}, \dots, o_0)$
	where $o_k = (\llbracket \mathsf{pd}(t) \rrbracket_\sigma)_k = \begin{cases} (\llbracket t \rrbracket_\sigma)_0 \vee (\llbracket t \rrbracket_\sigma)_1 & k = 0 \\ (\llbracket t \rrbracket_\sigma)_{k+1} & o.w. \end{cases}$
$\sigma(\lvert T \rvert) = \sigma(T)$	$\llbracket \lvert T \rvert \rrbracket_\sigma = \llbracket \sigma(T) \rrbracket_\sigma$
$\sigma(f(\vec{t}, \vec{T})) = f^\sigma(\sigma(\vec{t}), \sigma(\vec{T}))$	$\llbracket f(\vec{t}, \vec{T}) \rrbracket_\sigma = (f_{\sigma(f(\vec{t}, \vec{T}))}(\llbracket \vec{t} \rrbracket_\sigma, \llbracket \vec{T} \rrbracket_\sigma), \dots, f_0(\llbracket \vec{t} \rrbracket_\sigma, \llbracket \vec{T} \rrbracket_\sigma))$
$\sigma(F(\vec{t}, \vec{T})) = F^\sigma(\sigma(\vec{t}), \sigma(\vec{T}))$	$\llbracket F(\vec{t}, \vec{T}) \rrbracket_\sigma = (F_{\sigma(F(\vec{t}, \vec{T}))-1}(\llbracket \vec{t} \rrbracket_\sigma, \llbracket \vec{T} \rrbracket_\sigma), \dots, F_0(\llbracket \vec{t} \rrbracket_\sigma, \llbracket \vec{T} \rrbracket_\sigma))$

Terms are translated recursively: a number term t is translated to a sequence of size $\sigma(t) + 1$, and a string terms T is translated to a sequence of size $\sigma(T)$. A unary number n is represented by $\top\bot^n$, i.e. the a sequence of size $n + 1$ where only the nth bit[19] is $\top$. We can use any reasonable AC^0 formula for the translation of the functions of $\mathcal{L}_2$. The main requirement is that the translation of the axioms in io2Basic must have simple propositional proofs. The translation distributes over sequences, i.e. $\llbracket t_k, \dots, t_0 \rrbracket_\sigma = \llbracket t_k \rrbracket_\sigma, \dots, \llbracket t_0 \rrbracket_\sigma$.

▶ **Definition 10** (Translation of terms). Let σ be a translation context. The translation for terms is given in table 2. For any term t, $\mathsf{size}(\llbracket t \rrbracket_\sigma)$ is bounded by a polynomial in σ. If a variable x occurs in t, then $\mathsf{size}(\llbracket t \rrbracket_\sigma) \geq \sigma(x)$. In addition, $\mathsf{ldepth}(\llbracket t \rrbracket_\sigma) = O(1)$.

We can translate terms containing a string-valued function F which is defined by a monotone non-decreasing term p and a formula φ (see Section 2.2) as[20] $\lvert F(\vec{x}, \vec{X}) \rvert = p(\vec{x}, \lvert \vec{X} \rvert)$, and $y \in F(\vec{x}, \vec{X}) \leftrightarrow \varphi(\vec{x}, \vec{X}, y)$, using $\sigma(F(\vec{t}, \vec{T})) = p(\sigma(\vec{t}), \sigma(\vec{T}))$, and $(\llbracket F(\vec{t}, \vec{T}) \rrbracket_\sigma)_k = \llbracket \varphi(\vec{t}, \vec{T}, y) \rrbracket_{\sigma[y \mapsto k]}$. Similarly, for number-valued functions f defined by $f(\vec{x}) \leq p(\vec{x})$, and $f(\vec{x}) = y \leftrightarrow \varphi(\vec{x}, y)$ we can use $\sigma(f(\vec{t})) = p(\sigma(\vec{t}))$, $(\llbracket f(\vec{t}) \rrbracket_\sigma)_k = \llbracket \varphi(\vec{t}, y) \rrbracket_{\sigma[y \mapsto k]}$.

For example, if we include substring function $T[s, t]$ in the language defined in Section 2.1, we can translate it using $\sigma(T[s, t]) = \sigma(t)$ and $(\llbracket T[s, t] \rrbracket_\sigma)_k = \llbracket x < t \wedge s + x \in X \rrbracket_{\sigma[x \mapsto k]}$.

Formulas are also translated recursively: atomic formulas are translated directly to AC^0 formulas such that the axioms about them have simple propositional proofs. Logical connective are translated to themselves. Bounded number quantifiers are translated to $\bigwedge$ and $\bigvee$. Bounded string quantifiers are translated to propositional quantifiers. The number of quantified propositional variables will be equal to the bound.

▶ **Definition 11** (Translation of formulas). Let σ be a translation context. The translation of bounded formulas is given in table 3. For any formula φ, $\mathsf{size}(\llbracket \varphi \rrbracket_\sigma)$ is bounded by a polynomial in σ. If a variable x occurs freely in φ, then $\mathsf{size}(\llbracket \varphi \rrbracket_\sigma) \geq \sigma(x)$. Also $\mathsf{ldepth}(\llbracket \varphi \rrbracket_\sigma) = O(1)$ and $\mathsf{qdepth}(\llbracket \varphi \rrbracket_\sigma) = O(1)$. The number of quantified propositional variables is determined by the translation of the bounding terms for the string quantifiers.

[19] The index of the rightmost bit is 0.
[20] For the translation to work we need the size of the functions to depend only on the size of their inputs. If we want to include in our translation functions like msb whose size is not determined by the size of its inputs we need to use a language that has a length operation for numbers.

Table 3 Translation of Formulas

$\llbracket s = t \rrbracket_\sigma = \bigvee_{i \leq \sigma(s), \sigma(t)} (\llbracket s \rrbracket_\sigma)_i \wedge (\llbracket t \rrbracket_\sigma)_i$ $\llbracket s \leq t \rrbracket_\sigma = \bigvee_{i \leq \sigma(s)} \bigvee_{i \leq j \leq \sigma(t)} (\llbracket s \rrbracket_\sigma)_i \wedge (\llbracket t \rrbracket_\sigma)_j$ $\llbracket t \in T \rrbracket_\sigma = \bigvee_{i \leq \sigma(T)} (\llbracket T \rrbracket_\sigma)_i \wedge (\llbracket t \rrbracket_\sigma)_i$	$\llbracket \bot \rrbracket_\sigma = \bot$ $\llbracket \top \rrbracket_\sigma = \top$ $\llbracket \neg\varphi \rrbracket_\sigma = \neg\llbracket \varphi \rrbracket_\sigma$ $\llbracket \psi \wedge \varphi \rrbracket_\sigma = \llbracket \psi \rrbracket_\sigma \wedge \llbracket \varphi \rrbracket_\sigma$ $\llbracket \psi \vee \varphi \rrbracket_\sigma = \llbracket \psi \rrbracket_\sigma \vee \llbracket \varphi \rrbracket_\sigma$ $\llbracket \exists x \leq t \ \varphi \rrbracket_\sigma = \bigvee_{i \leq \sigma(t)} \llbracket x \leq t \wedge \varphi \rrbracket_{\sigma[x \mapsto i]}$ $\llbracket \forall x \leq t \ \varphi \rrbracket_\sigma = \bigwedge_{i \leq \sigma(t)} \llbracket x \leq t \rightarrow \varphi \rrbracket_{\sigma[x \mapsto i]}$ $\llbracket \exists X = t \ \varphi \rrbracket_\sigma = \exists \llbracket X \rrbracket_\tau \ \llbracket X = t \wedge \varphi \rrbracket_\tau$ $\llbracket \forall X = t \ \varphi \rrbracket_\sigma = \forall \llbracket X \rrbracket_\tau \ \llbracket X = t \rightarrow \varphi \rrbracket_\tau$. where $\tau = \sigma[X \mapsto \sigma(t)]$

Recall that bounded string quantifiers of the form $\exists X \leq t \ \varphi$ and $\forall X \leq t \ \varphi$ are equivalent to $\exists y \leq t \ \exists X = y \ \varphi$ and $\forall y \leq t \ \forall X = y \ \varphi$ and can be translated as such.

Using induction on the structure of formulas we can prove that $\llbracket \varphi \rrbracket_\sigma$ is a tautology iff $\vec{x} = \sigma(\vec{x}), |\vec{X}| = \sigma(\vec{X}) \Rightarrow \varphi$ is true in the standard model.

4.2 Translation of Proofs

In this section, we provide a translation from proofs in the theory n^ε-ioV^∞ to polynomial-size n^ε-bdG_∞ and subexponential-size bdG_0 proofs. We will consider proofs in n^ε-ioV^∞ where $\varepsilon = \frac{1}{d} < 1$ is fixed.

▶ **Theorem 12** (Propositional Translation). *If $\varphi \in \Sigma_0^B$ is provable in n^ε-ioV^∞, then $\{\llbracket \varphi \rrbracket_{\vec{n}}\}_{\vec{n}}$ has polynomial-size n^ε-bdG_∞.*

Assume that n^ε-$\mathsf{ioV}^\infty \vdash \varphi$. If φ has no free variables its translation is a fixed formula and has a size $O(1)$ depth $O(1)$ proof and we are done. So assume that φ has at least one free variable.

Without loss of generality, we can assume that all free variables in φ have input type: if φ is provable then so is $\varphi[\vec{x}, \vec{X}/\vec{a}, \vec{A}]$, which has the same translation. We refer to $\vec{n} = \sigma(\vec{a}, \vec{A})$ as translation parameters where $\vec{a}$ and $\vec{A}$ are φ's free variables. Let $m_{\vec{n}} = \mathsf{size}(\llbracket \varphi \rrbracket_{\vec{n}})$.

Since φ has at least one free variable we have $m_{\vec{n}} = \mathsf{size}(\llbracket \varphi \rrbracket_{\vec{n}}) = \Omega(\vec{n})$. Therefore, if prove that some entity like the size of a proof for φ is bounded by a monotone non-decreasing function of $\vec{n}$, the bound will also apply with $\vec{n}$ replaced by $\vec{m}$. With this in mind, we will study proof size, proof depth, etc. in terms of the translation parameters $\vec{n}$.

The propositional translation has three steps. In the first step, we Skolemize conversion and comprehension axioms to remove the initial existential string quantifiers from them. We denote the Skolemized version of these axioms by adding a * superscript. The Skolem functions for comprehension axioms return output-type values while the Skolem function for the conversion axiom returns input-type values. In other words, the type of value returned by a Skolem function matches the type of quantified variable it is witnessing. Note that axioms determine the size of the functions symbols.

- $\mathsf{oiConv}_{\mathsf{str}}$: $\Rightarrow \exists B = a \ \forall z < a \ (z \in B \leftrightarrow y + z \in X)$,
- $\mathsf{oiConv}^*_{\mathsf{str}}$: $\Rightarrow |F^{\mathsf{oiConv}}(a, y, X)| = a \wedge \forall z < a \ (z \in F^{\mathsf{oiConv}}(a, y, X) \leftrightarrow y + z \in X)$.
- φ-CA: $\Rightarrow \exists Y = z \ \forall x < z \ (x \in Y \leftrightarrow \varphi(x, A))$,
- φ-CA^*: $\Rightarrow |F^{\varphi\text{-}\mathsf{CA}}(z, A)| = z \wedge \forall x < z \ \big(x \in F^{\varphi\text{-}\mathsf{CA}}(z, A) \leftrightarrow \varphi(x, A)\big)$.

Let's call the resulting theory $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty}$. Note that the theory still has the nice properties of the original theory. In particular, the size of an input-type term is still bounded by a linear function of the size of its input-type variables. We have:

▶ **Lemma 13** (Step 1). *If n^ε-$\mathsf{ioV}^\infty \vdash \varphi$ then $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty} \vdash \varphi$.*

Proof of Step 1. We only need to derive the axioms of n^ε-ioV^∞ in $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty}$. The original axioms are derivable from the Skolemized versions using a single application of the $\exists\mathsf{R}$. ◀

Next, we translate proofs in $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty}$ to proof families in H with non-logical axioms.

▶ **Lemma 14** (Step 2). *If $\pi : \widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty} \vdash \varphi$, then there is a H-proof $\{[\![\pi]\!]_{\vec{n}}\}_{\vec{n}}$ of $\{[\![\varphi]\!]_{\vec{n}}\}_{\vec{n}}$ using the translation of the axioms in $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty}$ as non-logical axioms. Moreover, $\mathsf{size}([\![\pi]\!]_{\vec{n}}) = O(\mathsf{poly}(\vec{n}))$, $\mathsf{ldepth}([\![\pi]\!]_{\vec{n}}) = O(1)$, $\mathsf{qdepth}([\![\pi]\!]_{\vec{n}}) = O(1)$, the proof contains only cuts over $\Sigma^q_\infty(n^\varepsilon)$ formulas, and the number of eigenvariables in each sequent is $O(n^\varepsilon)$.*

Proof of Step 2. We first convert the proof π into a proof in free-variable free-cut free normal form. The resulting proof only contains subformulas of φ and the axioms. Since φ does not have any string quantifier, all cut formulas with string quantifiers are subformulas of the Skolemized comprehension axiom. By lemma 15 below, we can bound the size of free input-type string variables in the proof by linear size terms in parameters of the proof $\vec{n}$. Therefore, all formulas in the proof are $\Sigma^B_\infty(n^\varepsilon)$.

We translate the proof to a propositional proof in H recursively starting from the end-sequent. The translation is straightforward. The rules in H correspond to rules in LK. The only interesting cases are the number quantifier rules which need to be replaced by rules for $\bigvee$ and $\bigwedge$. For $\exists\mathsf{L}$ and $\forall\mathsf{R}$, we extend the translation context to assign values to the eigen variable for all possible values less than the bound. All terms have at most polynomial size in their free variables. Therefore, we will construct a polynomial number of them.

For example, consider $\forall\mathsf{R}$. The sequent $\Gamma \Rightarrow \Delta, \forall x \leq t\ \psi$ is translated to $[\![\Gamma]\!]_\sigma \Rightarrow [\![\Delta]\!]_\sigma, \bigwedge_{i\leq\sigma(t)} [\![x \leq t \to \psi]\!]_{\sigma[x\mapsto i]}$. We recursively obtain the proofs for the translations of $\Gamma \Rightarrow \Delta, x \leq t \to \psi$ under translation contexts $\sigma[x \mapsto i]$ for all $i \leq \sigma(t)$, and use $\bigwedge\mathsf{R}$ to obtain

$$\frac{\Gamma \Rightarrow \Delta, x \leq t \to \psi}{\Gamma \Rightarrow \Delta, \forall x \leq t\ \psi}\,\forall\mathbf{R} \qquad \frac{\{[\![\Gamma]\!]_\sigma \Rightarrow [\![\Delta]\!]_\sigma, [\![x \leq t \to \psi]\!]_{\sigma[x\mapsto i]}\}_{i\leq\sigma(t)}}{[\![\Gamma]\!]_\sigma \Rightarrow [\![\Delta]\!]_\sigma, \wedge_{i\leq\sigma(t)} [\![x \leq t \to \psi]\!]_{\sigma[x\mapsto i]}}\,\bigwedge\mathbf{R}$$

The axioms are translated to non-logical quantified propositional axioms. It is easy to check that the depth and quantifier depth of the proof are $O(1)$ and size of the proof is $O(\mathsf{poly}(\vec{n}))$. Formulas in $\Sigma^B_\infty(n^\varepsilon)$ are translated to $\Sigma^q_\infty(n^\varepsilon)$ formulas, so cuts formulas are $\Sigma^q_\infty(n^\varepsilon)$. Each sequent in the translated proof is a translation of a first-order sequent. Therefore, the number of formulas in each sequent is constant and the total number of eigenvariables in each sequent is $O(n^\varepsilon)$. ◀

▶ **Lemma 15** (Linear Type Bounds). *The size of free input-type string variables in the proof can be bounded by linear terms in $\vec{n}$.*

Proof Idea. The proof is similar to Parikh's theorem. ◀

In the third step, we remove the function symbols and non-logical axioms from the proof to obtain a polynomial-size n^ε-bdG_∞ proof.

▶ **Lemma 16** (Step 3). *If $\widehat{n^\varepsilon\text{-}\mathsf{ioV}^\infty} \vdash \varphi$ then $\{[\![\varphi]\!]_{\vec{n}}\}_{\vec{n}}$ has a polynomial-size n^ε-bdG_∞ proof.*

Proof Step 3. Consider the proof obtained in step 2. To obtain a n^ε-bdG_∞-proof we need to provide

- polynomial-size explicit witnessing formulas for the function symbols, and
- polynomial-size n^ε-bdG_∞ proofs for the non-logical axioms.

Note that the translations of the axioms of io2Basic have simple polynomial-size bdG_0 proofs. The the translation of the induction axiom becomes

$$[\![0 \in X]\!]_\sigma, \bigwedge_{i \le \sigma(z)} [\![y \le z]\!]_{\sigma[y \mapsto i]} \to ([\![y \in X]\!]_{\sigma[y \mapsto i]} \to [\![y+1 \in X]\!]_{\sigma[y \mapsto i]}) \Rightarrow [\![z \in X]\!]_\sigma$$

which has a simple proof of polynomial size and bounded depth: We provide proofs for $[\![y+1 \in X]\!]_{\sigma[y \mapsto i]} \Rightarrow [\![y \in X]\!]_{\sigma[y \mapsto i+1]}$ and then combine these using $\vee$L to obtain the proof.

We use substring functions to witness the conversion axiom. To remove the conversion axiom, we replace $[\![F^{oiConv_{str}}(a, y, X)]\!]_n$ with $[\![X[y, a]]\!]_n$. The axiom becomes

$$\Rightarrow [\![|X[y, a]| = a]\!]_n \wedge \bigwedge_{i \le \sigma(a)} [\![x \in X[y, a] \leftrightarrow y + x \in X]\!]_{n,[x \mapsto i]}$$

which has a bdG_0-proof of polynomial size and bounded depth.

We will use the defining formula of the comprehension axioms to witness the comprehension function symbols. To remove the comprehension function symbols, we replace $([\![F^{\varphi\text{-}CA}(A)]\!]_\sigma)_n$ with $[\![\varphi(x, A)]\!]_{n,[x \mapsto i]}$. The comprehension axioms become

$$\Rightarrow \bigwedge_{i \le \sigma(t)} \big([\![\varphi(x, A)]\!]_{n,[x \mapsto i]} \leftrightarrow [\![\varphi(x, A)]\!]_{n,[x \mapsto i]}\big)$$

which have bdG_0 cut-free proofs of polynomial size. ◀

Finally, we expand the $\Sigma^B_\infty(n^\varepsilon)$ formulas to bounded-depth formulas of size $2^{O(n^\varepsilon)}$. As a result, we get size $2^{O(n^\varepsilon)}$ bdG_0 proofs.

▶ **Theorem 17** (Subexpoential-Size Bounded-Depth G_0 Proofs). *If $\{[\![\varphi]\!]_{\vec{n}}\}_{\vec{n}}$ has a polynomial-size n^ε-bdG_∞ proof then $\{[\![\varphi]\!]_{\vec{n}}\}_{\vec{n}}$ has a size $2^{O(n^\varepsilon)}$ bdG_0 proof.*

Proof of Corollary 17. We convert the proof by replacing propositional quantifiers with $\bigwedge$ and $\bigvee$ and quantifier introduction rules by their $\bigwedge$ and $\bigvee$ counterparts. The result is a valid proof with no quantifiers. The depth of the formulas in the proof is still $O(1)$. Sine the number of quantified variables in any formula was $O(n^\varepsilon)$ the size of new formulas is $2^{O(n^\varepsilon)}$. Moreover, since the number of eigen variables in each sequent is $O(n^\varepsilon)$, we only need to make $2^{O(n^\varepsilon)}$ copies of them in total for replacing the quantifier introduction rules. Therefore, the size of the resulting proof is $2^{O(n^\varepsilon)}$. ◀

5 From Nonuniform to Uniform

The other half of the relation between a bounded arithmetic theory and a propositional proof system is given by the provability of the soundness of the propositional proof system (or proof class) inside the corresponding theory. The soundness statement for a proof system Q states that for every φ, π, and τ, if π is a Q-proof of the formula φ, and τ is a truth assignment for φ, then τ satisfies φ: $\forall \varphi, \pi, \tau\ (\pi : \mathrm{Q} \vdash \varphi \Rightarrow \tau \vDash \varphi)$. For a proof class that is obtained from taking the union of an indexed family of proof systems the soundness statement is defined as the set of soundness statements for each proof system in the family. For example, we say that a theory proves the the soundness of $\mathsf{bdFrege}$ iff it proves $\{Sound(d\text{-}\mathsf{Frege}) \mid d \in \mathbb{N}\}$.

The importance of soundness statements are their universal role in proof complexity similar to complete problems in complexity theory, i.e. proof classes can be characterized as the set of tautology families derivable from the soundness tautology families in a weak propositional proof system like resolution.

Let T be a theory extending ioV^0 and F be a proof class containing polynomial-size $\mathsf{bdFrege}$ and closed under cuts over $\mathsf{bd}\Sigma_0^{\mathsf{q}}$ formulas. Assume that F proves the translation of Σ_0^B theorems of T. We have

▶ **Theorem 18.** *If* T *proves the soundness of* F'*, then* $\mathrm{F}' \subseteq \mathrm{F}$.

Proof. First, if $\{\pi_n\}_n : \mathrm{F}' \vdash \{\varphi_n\}_n$, then this family has a polynomial-size $\mathsf{bdFrege}$ proof. Second, $\{(\tau \vDash \varphi_n) \Rightarrow \varphi_n(\tau)\}_n$ has a polynomial-size $\mathsf{bdFrege}$ proof. Now, since the soundness of F' is provable in T, the translation of the soundness of F', $\{\pi_n \vdash \varphi_n \Rightarrow \tau \vDash \varphi_n\}_n$, is provable in F. F is closed under $\mathsf{bd}\Sigma_0^q$ cuts, and proves $\{\pi_n : \mathrm{F}' \vdash \varphi_n\}_n$, therefore $\{\tau \vDash \varphi_i\}_i$ is provable in F. Which means $\{\varphi_n(\tau)\}_n$ is provable in F. To make the argument complete, we need discuss the accepting computations of the proof system containing F and the evaluation of formulas. See [13, 12] for details. ◀

▶ **Theorem 19.** *Let* F *be a* Q*-proof class satisfying conditions mentioned above. If the soundness of a proof system* Q' *is provable in* F*, then* Q *simulates* Q' *with* F *proofs. If the* F*-proofs are soundness of* Q' *are effectively given, then the simulation is effective.*

Proof. The proof is similar to the proof of Theorem 18. ◀

5.1 ioV^0 Proves Soundness of $\mathsf{bdFrege}$

▶ **Theorem 20** ([7])**.** *The soundness of the proof class* $\mathsf{bdFrege}$ *is provable in* V^0.

In soundness statements, the proof is given to us as an input. Since the size of the formulas in the proof are bounded by the size of the proof, we can easily evaluate these formulas in ioV^0 using the comprehension axiom. Similarly, the size of the proof is an input-type number, so we can use the induction axiom to prove that all sequents in the proof are true under a given assignment. At no point in the argument do we need to compute large values, so the provability of soundness of $\mathsf{bdFrege}$ works in ioV^0. Therefore

▶ **Theorem 21.** *The soundness of* $\mathsf{bdFrege}$ *is provable in* ioV^0.

5.2 ioVNC^1 Proves Soundness of (Unbalanced) Frege

The comprehension axiom of ioVNC^1 can be used to evaluate balanced formulas and therefore ioVNC^1 can prove the soundness of Frege proofs where formulas in the proof are balanced.

▶ **Theorem 22.** $\mathsf{ioVNC}^1 \vdash \mathit{Sound}(\mathsf{BalancedFrege})$.

But that does not necessarily imply that ioVNC^1 can prove the soundness of (unbalanced) Frege proofs. We need to balance formulas and provably so in ioVNC^1. It turns out that ioVNC^1 can prove the Buss's result [3, 4] that (unbalanced) Boolean formulas can be evaluated in $\mathsf{ALogTime}$ (which is equivalent to uniform NC^1). The Buss's proof [4] is formalized in VNC^1, see [7, pp. 410-424].

▶ **Theorem 23.** ioVNC^1 *proves the totality and correctness of Buss's* $\mathsf{ALogTime}$ *algorithm [4] for unbalanced Boolean formula evaluation problem.*

Proof. The goal is to prove that we can evaluate unbalanced formulas, i.e. for a formula and a truth assignment given in A and B, there is a Y which is a computation of A on B. Note that the computation doesn't need to encode the values obtained for gates, the evaluation is correct, i.e. it distributes over logical operations and correctly computes the value of $\top$

and $\bot$. We can use an AC^0 function to build a balanced formula Z from A using Buss's algorithm, and then apply MBBFE to Z and obtain a computation Y of it. Note that the $\Sigma_0^B(\mathsf{MBBFE})$-$\mathsf{CA}$ axiom allows this. The game tree of Buss's algorithm only depends on the size of the formula. The part of the balanced formula that depends on the formula is a TC^0 functions that given a game play and a formula decides the winner. We attach a balanced Boolean formula computing this TC^0 function to the leaves of the game tree.

Note that for correctness we don't need to compute any global function of output-type objects. So the argument in [7] still works. ◀

▶ **Corollary 24.** *The (unbalanced) Boolean formula evaluation is provably total in* ioVNC^1.

Now, following the standard argument we can prove the soundness of (unbalanced) Frege in ioVNC^1.

▶ **Theorem 25.** ioVNC^1 *proves the soundness of (unbalanced)* Frege

Proof. Let $\pi : \mathsf{Frege} \vdash \varphi$ be a Frege proof of φ. We show that φ is true. Let τ be an arbitrary truth assignment for the formulas in π. We show by induction on the size of π that the sequents in π are true under τ. For the base case, we have to verify that the axioms are true which is straight forward. For the induction step, we have to show that the rules preserver truth of sequents. The correctness of all rules can be verified by case analysis. We use Theorem 24 to show the correctness of the cut rule. ◀

5.3 n^ε-ioV^∞ Proves Soundness of n^ε-bdG_∞

First note that as a corollary of Theorem 21 we have

▶ **Corollary 26.** *The soundness of* $\mathsf{bdFrege}$ *is provable in* n^ε-ioV^∞.

▶ **Theorem 27.** *The soundness of* n^ε-bdG_∞ *is provable in* n^ε-ioV^∞.

Proof Idea. The argument follows the same structure of provably of soundness results. ◀

6 $\mathsf{ioVNC}^1 \subseteq n^\varepsilon$-$\mathsf{ioV}^\infty$

In this section, we prove that n^ε-$\mathsf{ioV}^\infty \vdash \mathsf{ioVNC}^1$ which is essentially formalizing and proving correctness of $\mathsf{NC}^1 \subseteq \mathsf{AltTime}(\mathsf{O}(1), \mathsf{O}(\mathsf{n}^\varepsilon))$. The argument also applies to other nice classes in $\mathsf{NTimeSpace}(\mathsf{n}^{\mathsf{O}(1)}, \mathsf{n}^{\mathsf{o}(1)})$ like NL.

▶ **Theorem 28.** *The theories* n^ε-ioV^∞ *contain the theory* ioVNC^1.

Proof of Theorem 28. We only need to derive $\Sigma_0^B(\mathsf{MBBFE})$-$\mathsf{CA}$ in n^ε-ioV^∞. Our goal is to show that there is a $\Sigma_\infty^B(n^\varepsilon)$ formula $\psi(x, A)$ which provably gives the bit graph of the computation Y of circuit Z in $\Sigma_0^B(\mathsf{MBBFE})$-$\mathsf{CA}$. In other words, $\psi(x, A)$ iff the value computed for gate x of the circuit Z is one. Therefore by $\Sigma_\infty^B(n^\varepsilon)$-$\mathsf{CA}$ the computation of circuit Z exists and we are done. Let's fix φ and s in $\Sigma_0^B(\mathsf{MBBFE})$-$\mathsf{CA}$ Note that s is bounded by a polynomial term in the size of free variables of the formula.

We think of φ as representing the graph of an AC^0 function. Abusing the notation, we use $\varphi(A)$ to denote this function. Consider a given A of length n. We can prove the existence of the $Z = \varphi(A)$ using the comprehension axiom in ioV^0. Let;s assume that Z and x are given.

Formula ψ is similar to Buss's algorithm. We describe it as a game between two players with k rounds. The first player claims that the correct value of gate x is 1 while the second player claims that is not the case. We refer to them as P (Prover) and C (Challenger).

Let Z be a balanced Boolean formula of size s and x a gate in Z. We divide Z into k levels. This results in subformulas of size $O(s^{1/k})$. We look at each of these small subformulas as a possible round in the game. The game tree has depth k and branching $O(s^{1/k})$. Each of these small formulas can be described by a path from the root to it. We index the subformulas, their inputs, and their computation using sequences of number $w = (i_1, i_2, \cdots, i_l)$ where $0 \leq l < k$ and each number is less than $s^{1/k}$. For example, the subformula in the root is indexed as $Z_{()}$. The subformulas below it are index as $Z_{(1)}$, $Z_{(2)}$, ... Each round is played in one of these subformulas, starting at the root subformula. After each step we will move to one of the subformulas below the current one. The game is finished when we reach a leave.

The game starts by the P giving a computation of the top subformula including the inputs to that subformula. If the computation is not correct P loses. Otherwise C challenges one of the inputs to the subformula whose output is the challenged input for the previous subformula. The games continues by moving to that subformula. The game ends when we reach the original input bits. At which point we can check if all claims (the claimed values for gates in each subcircuit are consistent and the value of the output gate of each subcircuit is equal to the value of the challenged input bit of the upper subcircuit) by P are correct, in which case P wins. Otherwise C wins.

The player P represents existential string quantifiers of size $O(s^{1/k})$. The player C represents universal number quantifiers of size $O(s^{1/k})$. Note that given a circuit, an input, and a computation, the fact that computation is correct is expressible as a Σ_0^B formula. We have k blocks of these quantifier followed by a Σ_0^B formula that checks if the claims are correct: 1. for each subformula in a game; the computation given for that subformula is compatible with the gates for that subformula; 2. the value of root for each subformula is equal the value of the leaf challenged in the previous round by C; 3. gate x belongs to one of the subformulas in the game; 4. the value of gate x is 1. If we pick $k = \frac{\lg s}{\varepsilon \lg n}$ then $s^{1/k} \leq n^{\varepsilon}$ and ψ is a $\Sigma_\infty^B(n^\varepsilon)$ formula. Note that ϕ is a σ_0^B formula giving the bit graph of Z and we can easily replace membership in and length of Z by φ and s.

Now that we have defined our Σ_0^B formula for φ and s, we have to show that it gives the bits of the computation of Z according to the definition in Σ_0^B(MBBFE)-CA, i.e. the value for each gate should be compatible with the type of the gate and the values for its children. This is mainly case analysis: either the gate is inside a subformula, in which case it is the case, or it is on the boarder between two level, in which case it is assigned the same values in both subformula computations. This completes the proof. ◀

7 Simulation of Frege proofs with bdFrege proofs

In this section, we combine the results from previous sections to provide an alternative proof of [10] that Frege proofs can be simulated by bdFrege proofs with a $2^{O(n^\varepsilon)}$ increase in proof size , i.e. size $2^{O(n^\varepsilon)}$bdFrege simulates polynomial-size Frege. These results also hold for size $2^{O(n^\varepsilon)}$bdFrege replaced with size $2^{O(n^\varepsilon)}$bdG$_0$.

Combining n^ε-ioV$^\infty \vdash$ ioVNC^1 from theorem 28 and ioVNC$^1 \vdash$ *Sound*(Frege) from theorem 25 we obtain the following corollary:

▶ **Corollary 29.** *The proof class n^ε-ioV$^\infty$ proves the soundness of the* Frege *proof system.*

Now, by Theorems 18 and 19 we have:

▶ **Corollary 30.** *The proof class polynomial size n^ε-bdG$_\infty$ contains the proof class polynomial-size* Frege. *The proof system* G *effectively simulates proof system* Frege *by polynomial-size n^ε-bdG$_\infty$ proofs.*

As a corollary of Theorem 17 that size $2^{O(n^\varepsilon)}$ bdFrege contains n^ε-bdG$_\infty$ we get

▶ **Corollary 31.** *The proof class size* $2^{O(n^\varepsilon)}$bdFrege *contains the proof class polynomial-size* Frege*. Size* $2^{O(n^\varepsilon)}$bdFrege *simulates polynomial-size* Frege*. The proof system* bdFrege *effectively simulates* Frege *by proofs of size* $2^{O(n^{n^\varepsilon})}$.

Acknowledgements. We would like to thank Phuong The Nguyen for many helpful discussions during the last two years. The idea for a uniform version of [10] came up during a discussion with Yuval Filmus, Toniann Pitassi, and Rahul Santhanam. We would like to thank the anonymous referees for their pointing out several sections which were unclear, and for their detailed and constructive comments and suggestions. The proofs of some lemmas are omitted because of length restrictions. Some sections of the current version may need further clarification. An earlier version of this work was presented in Fall 2011 by the first author while attending MALOA's special semester on Logic and Complexity in Prague. He would like to thank the organizers and participants, in particular Jan Krajíček, for creating a lovely environment for research and exchange of ideas.

References

1 Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3):14:1 – 14:36, 2010.

2 Samuel R. Buss. *Bounded Arithmetic.* Bibliopolis, 1986.

3 Samuel R. Buss. The boolean formula value problem is in alogtime. In Alfred V. Aho, editor, *STOC*, pages 123–131. ACM, 1987.

4 Samuel R. Buss. Algorithms for boolean formula evaluation and for tree-contraction. In P. Clote and J. Krajicek, editors, *Proof Theory, Complexity, and Arithmetic*, pages 95–115. Oxford University Press, 1993.

5 Samuel R. Buss and Ryan Williams. Limits on alternation-trading proofs for time-space lower bounds. In *IEEE Conference on Computational Complexity*, pages 181–191. IEEE, 2012.

6 Stepehn A. Cook. Feasibly constructive proofs and propositional calculus. In *Annual ACM Symposium on Theory of Computing*, volume 7, pages 83–97, 1975.

7 Stepehn A. Cook and Phoung Nguyen. *Logical Fouondations of Proof Complexity.* Cambridge University Press, 2010.

8 Stephen Cook and Tsuyoshi Morioka. Quantified propositional calculus and a second-order theory for NC1. *Archive for Mathematical Logic*, 44(6):711–749, 2005.

9 Stephen A. Cook. Relativized propositional calculus. Manuscript, 2012.

10 Y. Filmus, T. Pitassi, and R. Santhanam. Exponential lower bounds for ac-frege imply superpolynomial frege lower bounds. *Proceedings ICALP*, 2011:618–629, 1.

11 Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Annals of Pure and Applied Logic*, 129:1–37, 2004.

12 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

13 Jan Krajíček. A note on sat algorithms and proof complexity. *Information Processing Letters*, 112(12):490–493, June 2012.

14 Sebastian Müller. Polylogarithmic cuts in models of V^0. *LMCS*, 9:2013, 2013.

15 V.A. Nepomnjascij. Rudimentary predicates and turing calculations. *Doklady AN SSSR*, 195, 1970.

16 J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In CarlosAugusto Prisco, editor, *Methods in Mathematical Logic*, volume 1130 of *Lecture Notes in Mathematics*, pages 317–340. Springer Berlin Heidelberg, 1985.