Uniformity and Nonuniformity in Proof Complexity

by

Kaveh Ghasemloo

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

© Copyright 2016 by Kaveh Ghasemloo

Abstract

Uniformity and Nonuniformity in Proof Complexity

Kaveh Ghasemloo
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2016

This thesis is dedicated to the study of the relations between uniform and nonuniform proof complexity and computational complexity. Nonuniform proof complexity studies the lengths of proofs in various propositional proof systems such as Frege. Uniform proof complexity studies the provability strength of bounded arithmetic theories which use only concepts computable in specific computational complexity classes, e.g. the two-sorted bounded arithmetic theory VNC^1 uses only concepts computable in NC^1.

We are interested in transferring concepts, tools, and results from computational complexity to proof complexity. We introduce the notion of *proof complexity class* which corresponds to the notion of computational complexity class. We show the possibility of developing a systematic framework for studying proof complexity classes associated with computational complexity classes. The framework is based on *soundness statements* for proof complexity classes and evaluation problems for circuit complexity classes. The soundness statements are *universal* for proof complexity classes as circuit evaluation problems are complete for computational complexity classes.

We introduce the notion of *io-typed theories* to design theories corresponding to computational complexity classes which are not closed under composition. We use io-types to control the composition of provably total functions of theories. We design a new class of theories n^{ε}-ioV^{∞} ($\varepsilon < 1$) corresponding to $\mathsf{AC}^0(\exp(n^{\varepsilon})) = \mathsf{AltTime}(O(1), O(n^{\varepsilon}))$, bounded-alternation sublinear-time computable concepts. This is the uniform counterpart of subexponential-size bounded-depth circuits. We provide a propositional

translation from the proofs of Σ_0^B-theorems in n^ε-ioV^∞ to subexponential-size bounded-depth proof families in Frege.

We prove that $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ contains $\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)})$. This implies that Boolean formulas can be evaluated in $\mathsf{AltTime}(O(1), O(n^\varepsilon))$. We formalize this containment inside our n^ε-ioV^∞ theories and prove that n^ε-ioV^∞ for $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ contains ioVNC^1 for NC^1. This is the proof complexity version of $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ containing NC^1 and a uniform version of the nonuniform proof complexity result that polynomial-size Frege proofs can be transformed to subexponential-size bounded-depth Frege proofs [FPS15].

Finally, we combine the universality of soundness, the propositional translation, and the containment $\mathsf{ioVNC}^1 \subseteq n^\varepsilon$-$\mathsf{ioV}^\infty$ to obtain an alternative proof of [FPS15].

Dedication

My heart ordered for thee: "Seek my face."
Thy face, my Lord, I seek.

Psalm 27 of David

Acknowledgements

First and foremost I would like to express my deepest gratitude to my adviser Stephen A. Cook for his unwavering support over the years, for his patience with my wanderings, for his careful critique of my ideas, for his reading of countless drafts of this thesis over the past year, and for his kindness in general towards me. Steve and his wife dear Linda made me feel part of their family.

I wish to thank the members of my supervisory committee Toniann Pitassi, Alasdair Urquhart, and Charles Rackoff for their valuable feedback over the years. The central questions around which this thesis is built came up during discussions with Toni. It is a pity that wise words often reveal their true value to young and rash only years later, if I could go back I would take her kindly-expressed critique of my work twice more seriously. For me Steve and Toni are the archetypes of the ideal scientist. The joy of having Alasdair on my committee is inexpressible. I have been pleasantly humbled numerous times by his vast knowledge of history and philosophy of mathematics. Charlie kindly agreed to serve on my committee despite the distance of my research from his. I am thankful for his always-frank critique of my work.

Thanks to the members of my non-supervisory examination committee, Jan Krajíček, Yashar Ganjali, and Faith Ellen. Faith kindly agreed to serve as a reserve member on a very short notice. Yashar kindly accepted to read my thesis despite its topic being rather remote from his research interests. I was honored to have Jan as my external examiner. His deep understanding of proof complexity is beyond mere mortals, open questions for others are easy consequences from where he stands. I cannot thank Jan enough for reading my thesis with extraordinary diligence and providing several suggestions for improvement in his appraisal.

I would like to thank Jan also for organizing proof complexity fall schools in their beautiful city – Prague. I learned a lot from him and other members of the Prague proof complexity group, Pavel Pudlák, Neil Thapen, and Emil Jeřábek. Jan, Pavel, and Sam Buss have been my proof complexity go-to experts outside Toronto and have kindly answered my questions for years. I am indebted to Phuong The Nguyen who provided valuable feedback in various stages of my work, shared with me his unpublished drafts on witnessing theorems, and pointed me to Steve's old unpublished work on QPC(R). Many thanks to Antonina Kolokolova for inviting me to Proof Complexity workshop at Banff International Research Station (BIRS) in 2011. Thanks to Jakob Nordström for inviting me to Theoretical Foundations of Applied SAT Solving workshop at BIRS in 2014. Both workshops were truly exceptional experiences where I enjoyed many exciting discussions with other participants. Thanks to my friends and colleagues in

the proof complexity community with whom I have enjoyed many adventures and discussions: Anupam Das, Yuval Filmus, Luke Friedman, Dai T.M. Le, Massimo Lauria, Sebastian Müller, and Ján Pich.

I was lucky to be part of Department of Computer Science at University of Toronto. Thanks to faculty members whose courses I have enjoyed immensely. Thanks to faculty and staff members at our department for creating a wonderful work environment. In particular I would like to thank Sara Burns, Lisa DeCaro, Marina Haloulos, and Sarah Lavoie in the main office. Special thanks to Sam Toueg who is a master in the art of teaching well; co-teaching with him was a truly transformational experience for me.

I do not know if I would have survived the years of my Ph.D. without my friends in Toronto and fellow students at our department. I wish I could *list them all* but they are simply too many to list here, nonetheless I am thankful to them all.

Toronto has become my home because of my dear friends with whom I have spent most of my Friday nights and weekends. Again too many to list and thank individually and this longer-than-usual acknowledgment would become way longer if I start to list friends outside Toronto whom I should thank, specially those in the Bay Area who welcomed me among themselves during my internship at Google in 2014 and 2015. The most valuable thing I have acquired over the course of my Ph.D. is many wonderful friends for whom I am deeply grateful. I simply cannot avoid thanking Amirhossein Shokouh Aghaei, Safa Akbarzadeh, Ali Ashasi Sorkhabi, Afshar Ganjali, and Ahmad Sobhani who were there when I needed them.

Finally, thanks to my siblings Kiarash, Kamyar, and Kimia for their unshakable love; my mother Mahlegha Vaheddoost who never stops worrying about us; and my father Taher Ghasemlou who was my first math and programming teacher and because of whom I got interested in math and computers.

Contents

List of Tables

List of Figures

List of Symbols

Notation	Description	Page
	Conventions	
A, B, C	input-type string variables	18
a, b, c	input-type number variables	18
d, i, j, k, l	natural numbers	
F, G	string-valued functions in the standard model	18
f, g	number-valued functions in the standard model	18
M, N	strings in the standard model	20
m, n	numbers in the standard model	20
p, q, r	propositional variables	30
S, T	string terms	18
s, t	number terms	18
X, Y, Z	output-type string variables	18
x, y, z	output-type number variables	18
α, β, γ	propositional function variables	30
δ, ε	positive real numbers	
Γ	set of formula families	
$\Gamma, \Delta, \Sigma, \Pi$	collections (sets, finite sequences) of formulas	18, 30
μ	complexity measure (size, logical depth, etc.)	32
Φ	class of formulas	
φ, ψ	formulas	18, 30
π	proof	20, 33
σ	translation context	45
τ	truth assignment, evaluation context	31
$\mathcal{C}$	formula complexity class	32
$\mathcal{F}$	proof complexity class	36
$\mathcal{L}$	language	
$\mathcal{M}$	model	
$\mathcal{Q}$	proof system	33
$\mathcal{R}$	proof complexity reduction class	38
$\mathcal{S}$	sequent	18, 31
$\mathcal{T}$	set of sequents, theory	18
$\mathcal{V}$	set of variables	30

Notation	Description	Page
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	placeholders for syntactic expressions	

Axioms and Computational Tasks

BFE	Boolean formula evaluation, see [Bus87; Bus93; AK10]	
BQCon	consistency statement, see [Bus86]	
φ-CA	comprehension axiom schema	25
$\mathcal{C}$-Eval, Eval($\mathcal{C}$)	evaluation for circuit complexity class $\mathcal{C}$	15
CV	circuit value	
$\mathcal{F}$-Snd, Snd($\mathcal{F}$)	soundness of proof complexity class $\mathcal{F}$	40, 57
Ind	induction axiom schema	25
MBBFE	monotone balanced Boolean formula evaluation	70
mod_p	counting mod p	
oiConv	conversion axioms	26
PHP	pigeonhole principle	32
SAT	the propositional satisfiability problem	

Complexity Classes

AC^0	$\mathsf{AltTime}(O(1), n^{O(1)})$, equals DLogTime-uniform polynomial-size bounded-depth unbounded-fan-in Boolean circuits	
AC^0_d	$\mathsf{AltTime}(d, n^{O(1)})$, equals DLogTime-uniform polynomial-size depth-d unbounded-fan-in Boolean circuits	
$\mathsf{AC}^0[p]$	AC^0 with mod_p gates	
$\mathsf{AC}^0/\mathsf{poly}$	AC^0 with a polynomial amount of advice, nonuniform AC^0, polynomial-size bounded-depth unbounded-fan-in Boolean circuits	
$\mathsf{AC}^0(R(n))$	AC^0 with n in resource bounds replace with $R(n)$	61
ALogTime	logarithmic-time alternating Turing machines, equals AC^0	
AltTime	alternating Turing machines with alternation and time bounds	
BPP	bounded-error probabilistic polynomial time	
CNF	polynomial-size CNF formulas	
coNP	conondeterministic polynomial-time Turing machines	
DepthSize	Boolean circuits with depth and size bounds	
DLogTime	deterministic logarithmic time	
DNF	polynomial-size DNF formulas	
DTime	deterministic Turing machines with time bounds	
ExpF	exponential-time deterministic Turing machines with no size bound on output	
k-ExpF	k-fold-exponential-time deterministic Turing machines with no size bound on output	
FO	first-order queries, equals AC^0, see [Imm99]	
FP	functions in deterministic polynomial-time Turing machines	

Notation	Description	Page
L	deterministic logarithmic-space Turing machines	
LH	logarithmic-time hierarchy, equals AC^0	
NC^1	the first level of Nick's class NC, $\mathsf{AltSpace}(O(\lg n), O(\lg n))$, equals DLogTime-uniform polynomial-size logarithmic-depth bounded-fan-in Boolean circuits, DLogTime-uniform polynomial-size Boolean formulas	
$\mathsf{NC}^1/\mathsf{poly}$	NC^1 with a polynomial amount of advice, nonuniform NC^1, polynomial-size logarithmic-depth bounded-fan-in Boolean circuits, polynomial-size formulas	
NC^k	$\mathsf{AltSpace}(O(\lg^k n), O(\lg n))$	
$\mathsf{NC}^k(R(n))$	NC^k with n in bounds replaced with $R(n)$	61
NL	nondeterministic logarithmic-space Turing machines	
NP	nondeterministic polynomial-time Turing machines	
$\mathsf{NTimeSpace}$	nondeterministic Turing machines with time and space bounds	
P	deterministic polynomial-time Turing machines	
PH	polynomial-time hierarchy	
PP	probabilistic polynomial-time Turing machines	
P/poly	P with a polynomial amount of advice, nonuniform P, polynomial-size Boolean circuits	
PSpace	polynomial-space Turing machines	
SC	Steve's class, $\mathsf{TimeSpace}(n^{O(1)}, \lg^{O(1)} n)$	
Size	Boolean circuits with size bound	
SO	second-order queries, equals PH, see [Imm99]	
TC^0	AC^0 with threshold/majority gates	
$\mathsf{TimeSpace}$	deterministic Turing machines with time and space bounds	

Formula Classes

$\exists^B \Sigma_0^B$	formulas consisting of $\exists^B$ quantifiers followed by a Σ_0^B formula, corresponds to NP	24
Σ_0^B	first-order formulas with only bounded number quantifiers, corresponds to AC^0	19
Σ_∞^B	first-order formulas with bounded number and string quantifiers, corresponds to PH	19
$\Sigma_\infty^{B \le t(n)}$	Σ_∞^B formulas where string quantifiers are bounded by $O(t(n))$ terms in the size of its free variables, corresponds to $\mathsf{AltTime}(O(1), O(t(n)))$	68
$\Sigma_\infty^{B \le n^\varepsilon}$	$\Sigma_\infty^{B \le t(n)}$ with $t(n) = n^\varepsilon$, Σ_∞^B formulas where string quantifiers are bounded by $O(n^\varepsilon)$ terms in the size of its free variables, corresponds to $\mathsf{AltTime}(O(1), O(n^\varepsilon))$	

Notation	Description	Page
Σ_0^q	quantifier-free propositional formulas	30
Σ_∞^q	quantified propositional formulas with a bounded number of quantifier alternations	30
$\Sigma_\infty^{q\leq t(n)}$	Σ_∞^q formulas with a bounded number of alternations (including AND and OR gates) where the number of quantified variables is bounded by $t(n)$	69
$\Sigma_\infty^{q\leq n^\varepsilon}$	$\Sigma_\infty^{q\leq t(n)}$ with $t(n) = n^\varepsilon$, Σ_∞^q formulas with a bounded number of alternations (including AND and OR gates) where the number of quantified variables is bounded by $O(n^\varepsilon)$	

Proof Systems and Classes

bdFrege	Frege proof families with cuts over bounded-depth propositional formulas	34
bdG	G proof families with cuts over formulas with a bounded-depth quantifier-free part	34
bdG_0	G_0 proof families with cuts over formulas with a bounded-depth quantifier-free part	34
bdG_∞	G_∞ proof families with cuts over formulas with a bounded-depth quantifier-free part	34
$t(n)$-bdG_∞	bdG_∞ proof families with at most $O(t(n))$ eigenvariables	69
n^ε-bdG_∞	$t(n)$-bdG_∞ with $t(n) = n^\varepsilon$, bdG_∞ proof families with at most $O(n^\varepsilon)$ eigenvariables	
CFrege	the circuit Frege proof system, equivalent to EFrege, see [Jeř05]	
Cut	the cut rule	34
EFrege	the extended Frege proof system, equivalent to CFrege	
Ext	the extensionality rule for propositional function symbols	35
Frege	the Frege proof system, defined as PK	34
G	quantified propositional proof system, obtained from PK by adding propositional quantifier introduction rules	34
G_0	G with cuts restricted to quantifier-free formulas	34
G_∞	G proof families with cuts over formulas with a a bounded number of quantifier alternations	34
H	relativized quantified propositional proof system, obtained from G by allowing propositional function symbols and the Ext rule	35
LK	first-order sequent calculus	20
PK	propositional sequent calculus	34
$\mathcal{Q}$-DepthSize	$\mathcal{Q}$ proofs with cut-formula depth and size bounds	37
$\mathcal{Q}$-Size	$\mathcal{Q}$ proofs with size bound	36
QPC(R)	relativized quantified propositional calculus, see [Coo12]	
Res	the resolution proof system	

Notation	Description	Page
	Theories	
2Basic	base two-sorted theory, see [CN10]	
$\mathsf{I}\Delta_0$	one-sorted theory with induction over Δ_0^b formulas	
$\mathsf{I}\Delta_0 + \mathsf{Exp}$	$\mathsf{I}\Delta_0$ plus exponentiation axiom	
io2Basic	io-typed base two-sorted theory	25
ioV^0	io-typed two-sorted theory for AC^0	26
ioVC	io-typed two-sorted theory for complexity class $\mathcal{C}$	27
ioVNC^1	io-typed two-sorted theory for NC^1	71
ioV^∞	io-typed two-sorted theory for $\mathsf{PH} = \mathsf{AltTime}(O(1), n^{O(1)})$	
$t(n)$-ioV^∞	io-typed two-sorted theory for $\mathsf{AltTime}(O(1), O(t(n)))$	68
n^ε-ioV^∞	$t(n)$-ioV^∞ with $t(n) = n^\varepsilon$, io-typed two-sorted theory for $\mathsf{AltTime}(O(1), O(n^\varepsilon))$	
ioVTC^0	io-typed two-sorted theory for TC^0	
PA	Peano arithmetic	
PV	equational theory for "feasible" reasoning, see [Coo75]	
RCA_0	second-order arithmetic theory with comprehension axiom for recursive sets, see [Sim09]	
S_2	bounded arithmetic theory, see [Bus86]	
V^0	two-sorted theory for AC^0, see [CN10]	
$\mathsf{V}\mathcal{C}$	two-sorted theory for complexity class $\mathcal{C}$, see [CN10]	
VNC^1	two-sorted theory for NC^1, see [CN10]	
VPV	two-sorted theory for P, see [CN10]	
VTC^0	two-sorted theory for TC^0, see [CN10]	
ZFC	Zermelo–Fraenkel set theory with axiom of choice	
	Miscellaneous	
$\mathcal{L}_2$	two-sorted language	18
$\mathbb{N}$	one-sorted standard model, the set of natural numbers	
$\mathbb{N}_2$	two-sorted standard model	19
num	sort of unary numbers	18
num^i	sort of input-type unary numbers	18
num^o	sort of output-type unary numbers	18
str	sort of binary strings	18
str^i	sort of input-type binary strings	18
str^o	sort of output-type binary strings	18
$\exp(x)$	2^x	
$x^{\frac{1}{d}}$	$\lfloor x^{\frac{1}{d}} \rfloor$	24
$x \in Y$	xth bit of Y is set	19
$\lvert Y \rvert$	length of string Y	19

Notation	Description	Page
$\mathsf{msb}(Y)$	most significant bit of Y	49
$\Rightarrow$	sequent	31
$Y = x$	length of string Y is x, e.g. $\forall Y = x$ quantifies over Ys of length x	19
$Y[s,t]$	substring of Y starting at s of length t	24
$\lvert\alpha\rvert$	the arity of the propositional function symbol α	30
$\ulcorner\mathcal{X}\urcorner$	a fixed encoding of $\mathcal{X}$ as a binary string	15
$(\mathcal{X})_i$	ith item in the sequence $\mathcal{X}$	45
$\mathcal{X}[\mathcal{Y}/\mathcal{Z}]$	substitution, occurrences of $\mathcal{Y}$ in $\mathcal{X}$ are replaced with $\mathcal{Z}$	31
$t(n)$	$t(n)$ is a number term in n, e.g. we use $t(n) = n^{\varepsilon}$ in chapter 8	67
"$\mathcal{X}$"	a fixed expressing of $\mathcal{X}$ as a formula	15
$[\![\mathcal{X}]\!]_\sigma$	propositional translation of $\mathcal{X}$ under σ	45, 47, 50
$\mathcal{M} \models \mathcal{T}$	$\mathcal{M}$ is a model of $\mathcal{T}$	
$\mathcal{T} \vdash \mathcal{S}$	$\mathcal{T}$ proves $\mathcal{S}$	
$\pi : \mathcal{Q} \vdash \mathcal{S}$	π is a $\mathcal{Q}$ proof of $\mathcal{S}$	33
$\tau \models \mathcal{S}$	τ satisfies $\mathcal{S}$	31

Chapter 1

Introduction

1.1 Motivation

The fundamental question of computational complexity is classifying computational tasks based on the amount of resources required to solve them in various models of computation [Edm65; HS65; Emd90; Vol99; CK02; FH03; AB09; Juk12]. The amount of resources for solving a computational task is typically measured as a function of the size of the input representing an instance of the computational task at hand. The models of computation themselves are divided into two main categories: uniform, e.g. various Turing machine models, where a single object encompasses a single algorithm solving any instance of the computational task; and nonuniform, e.g. various circuit models, where the computational task is solved by a family of independent programs, one for each input size. In other words, the specification of a program for solving task instances of size n can depend on n; while the specification of an algorithm is independent of n. We use "algorithm" to refer to a member of some uniform computational model and "program" to refer to a member of some nonuniform computational model. Nonuniform models are considered hardware computational solutions whereas uniform models are considered software solutions [KL82]. A nonuniform program for solving task instances of a particular size can hardcode a small amount of extra information which can help in solving those task instances. This extra information called *advice* is essentially the nonuniform part of the program [Sch76; Pip79; Ruz79; KL82]. Uniform algorithms are as capable as nonuniform programs generated by a uniform algorithm; while uniform algorithms with suitable advice are as capable as nonuniform programs. See figure 1.1. A uniform algorithm A can be considered a family of programs with no nonuniform advice by simply taking the family $\{A\}_n$

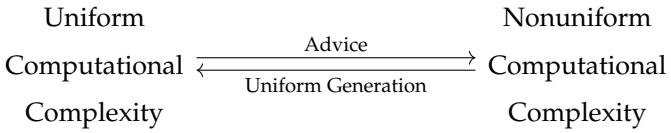


Figure 1.1: Uniformity and Advice for Computational Complexity

and interpreting the nth member of the family as the algorithm A restricted to task instances of size n.

A computational complexity class is typically defined by introducing measures of resource complexity for a model of computation and putting restrictions on the amount of available resources. E.g. the class of (decision problems solvable by) polynomial-time Turing machines $\mathsf{P} = \mathsf{DTime}(n^{O(1)})$ is obtained by defining the worst-case running-time of Turing machines and considering only those Turing machines which run in at most polynomial time with respect to the size of their input. Similarly, the class of (decision problems solvable by) polynomial-size circuit families $\mathsf{Size}(n^{O(1)})$ is obtained by defining the size of circuits and considering only circuit families of polynomial size with respect to the size of their input. Transformations from uniform to nonuniform and from nonuniform to uniform mentioned in the previous paragraph often remain valid for *robust* complexity classes with suitable *advice* and *uniformity* conditions. E.g. $\mathsf{DTime}(n^{O(1)})$ machines with a polynomial amount of advice are as capable as $\mathsf{Size}(n^{O(1)})$ circuit families; while $\mathsf{DTime}(n^{O(1)})$ machines are as capable as $\mathsf{Size}(n^{O(1)})$ circuit families uniformly generated by a $\mathsf{DTime}(n^{O(1)})$ algorithm. For this reason $\mathsf{Size}(n^{O(1)})$ is also denoted by $\mathsf{P/poly}$ — the class of (decision problems solvable by) polynomial-time Turing machines with a polynomial amount of advice. There are two general ingredients for these transformations: the transformation of a uniform algorithm with a fixed input size to a program as in Cook's theorem [Coo71]; and a uniform algorithm for evaluation of programs on inputs, e.g. there is an algorithm in P for the *Circuit Value Problem* [Lad75]. See figure 1.2. Our main reference for uniform and nonuniform computational complexity is [Vol99].

Proof complexity, in turn, is concerned with the study of the amount of "reasoning resources" required for proving mathematical theorems [Coo75; Rec76; CR79; Bus86; Kra95; CN10]. Proof complexity is mainly concerned with provability of universal formulas of the form $\forall n \ \varphi(n)$ in first-order arithmetic theories. Reasoning in mathematics is typically carried out in first-order theories like ZFC or PA, implicitly if not explicitly. The main topic of study in uniform proof complexity is the provability of universal formulas in various arithmetic theories. E.g. we can study the provability of

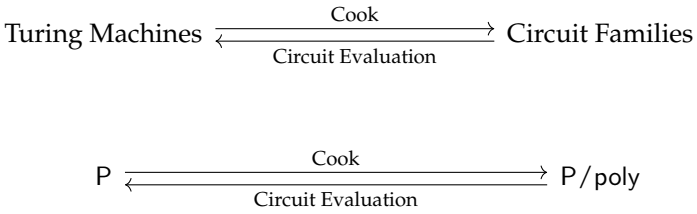


Figure 1.2: Uniform and Nonuniform Computational Models

the soundness of SAT algorithms uniformly in a theory like PV for polynomial-time computable concepts [Coo75; CU93]. We can view a universal formula $\forall n\ \varphi(n)$ as a family of propositional formulas $\{\varphi(n)\}_n$. The main topic of study in nonuniform proof complexity is the length of proofs for families of tautologies. E.g. we can study the length of nonuniform proofs for the soundness of SAT algorithms in a proof system restricted to polynomial-time computable concepts like EFrege where lines in the proof correspond to polynomial-size circuits [Kra12].

Early work in uniform proof complexity stemmed from the logical studies of arithmetic with concerns about computational feasibility [Par71; Coo75]. Nonuniform proof complexity originated from the study of the computational complexity of automatic theorem proving algorithms [Tse68; Coo71; CR74; Coo75]; which led to the theory of NP-completeness and the notion of polynomial-time computable reductions between problems. Proof systems can be viewed more generally as nondeterministic algorithms for the tautology problem TAUT, the dual of the NP-complete satisfiability problem SAT. An accepting execution history of a sound and complete algorithm for TAUT on a tautology can be viewed as a proof for the tautology. Similarly, a rejecting execution of a sound and complete algorithm for SAT on an unsatisfiable formula φ can be viewed as a refutation proof for the validity of $\neg\varphi$. The idea of viewing the execution history of an algorithm on an input as a proof goes back to at least the dawn of computational complexity theory, e.g. [Rab60]. The amount of reasoning resources used in a proof of a tautology is often closely related to the amount of computational resources used by SAT algorithms corresponding to the proof system, e.g. the length of a proof corresponds to the length of an execution history, which in turn is closely tied to its running-time. The existence of a *super* proof system where every propositional tautology has a polynomial-size proof is equivalent to NP $=$ coNP. We are not close to settling the existence of a super proof system. Even the question about the existence of an *optimal* proof system which up to a polynomial overhead is as efficient as any other proof system remains open [KP89; Kra95]. New links between the existence of an optimal proof system and open questions have been discovered in recent years. E.g.

the existence of an optimal proof system is equivalent to the existence of a descriptive logic that captures P in descriptive complexity theory [CF10].

The length of proofs in proof systems is also of interest for algorithmic reasons even when those proof systems are known not to be super. A rejecting execution of the DPLL SAT algorithm [DP60; DLL62] and its variants — including those using various heuristics like *Variable State Independent Decay* (VSID) variable branching [KSM11], *Conflict-Driven Clause Learning* (CDCL) [MLM09], and *random restarts* [GSK98] — can be converted to a proof in the Resolution proof system [Rob65] of similar size. As a result, an exponential lower bound on the length of the proofs of the Pigeonhole Principle tautologies (PHP) in a proof system like Resolution [Hak85] implies a lower bound on the running time of the DPLL-based SAT algorithm used in practice. In the other direction, it is known that variants of CDCL SAT algorithms where heuristics like VSID are replaced with nondeterminism can find proofs corresponding to proofs in Resolution and its subsystems [BKS04; PD11; BS14b]. These connections provide a theoretical foundation for evaluating and developing SAT algorithms supplementing the experimental benchmarking employed by engineers and practitioners for the evaluation and comparison of SAT algorithms. All known explicit hard instances for SAT algorithms are obtained through proof complexity lower bounds. Similar connections exist between semi-algebraic proof systems and optimization algorithms for linear, semi-definite, and sum of squares programming frameworks [GHP02; CT12; BS14a].

The soundness of a correct algorithm on inputs of a fixed size can be viewed as a tautology. Therefore, we can study the proof complexity of proving the soundness of algorithms. This provides a conceptual measure for classifying algorithms in addition to their computational complexity. Fascinatingly, soundness statements turn out to be universal for proof complexity classes analogous to circuit evaluation problems being universal for computational complexity classes. If a proof system $\mathcal{Q}$ has a short proof for the soundness of another proof system $\mathcal{Q}'$ and satisfies some basic closure properties, it has short proofs for all tautologies with short proofs in $\mathcal{Q}'$ [Coo75; Kra12]. *Soundness tautologies are analogous to complete problems for complexity classes.*

The classification of algorithms based on the axioms required to prove their soundness is a special case of *bounded reverse mathematics* [Ngu08; CN10], which itself is a refinement of *reverse mathematics* [Sim09] concerned with *the computational complexity of concepts* required for proving mathematical theorems rather than their computability. Reverse mathematics classifies theorems using RCA_0 as the base theory, which corresponds to the comprehension axiom for recursive sets. Bounded reverse mathematics

	Formula	Tautology
Uniform	$\varphi(X)$	$\vDash \forall X\ \varphi(X)$
Sectioned	$\{\varphi(X)\}_n$	$\vDash \{\forall X = n\ \varphi(X)\}_n$
Advice	$\{\varphi(X, F(n))\}_n$	$\vDash \{\forall X = n\ \varphi(X, F(n))\}_n$
Nonuniform	$\{\varphi_n(X)\}_n$	$\vDash \{\forall X = n\ \varphi_n(X)\}_n$

F is an advice function from natural numbers to binary strings.

Figure 1.3: Uniform and Nonuniform Tautologies

classifies theorems using V^0 as the base theory. The theory V^0 is a two-sorted version of Buss's bounded arithmetic theories [Bus86] based on the two-sorted language of Zambella [Lei91; Ign95; Zam96] and corresponds to the comprehension axiom for AC^0 sets [Ngu08; CN10]. Theories and proof systems have been designed for various computational complexity classes and to formalize results in computer science and mathematics in them [Sol01; Jeř05; Kol05; Mor05; Ske06; Pit07; Ngu08; Aeh10; Lê14; Pic14]. These theories are typically defined either using a descriptive complexity characterizations or a complete problem for the complexity class. Our main reference for descriptive complexity is [Imm99].

Bounded arithmetic theories have also been used as a framework to study and formalize the difficulty of proving lower bounds in computational complexity theory. The natural proofs barrier [RR94] for circuit lower bounds was conceived in tandem with attempts to formalize the existing circuit lower bounds in weak bounded arithmetic theories [Raz95a] (which resulted in a significantly simpler proof of Håstad's Switching Lemma) and prove that strong circuit lower bounds are not attainable in those theories [Raz95b; Pic15].

These bounded arithmetic theories correspond to uniform computational complexity classes and Cook-Reckhow proof systems correspond to nonuniform models of computation. There are connections between uniform and nonuniform proof complexity parallel to computational complexity theory. We can consider a proof π of a formula $\forall X\ \varphi(X)$ in a theory $\mathcal{T}$ as a family of proofs by fixing the size of the free variable X: $\{\pi(n) : \mathcal{T} \vdash \forall X = n\ \varphi(X)\}_n$. We can translate these formulas to (quantified) propositional formulas using propositional translation if $\varphi(X)$ is a bounded formula. Nonuniformity can be obtained by adding advice. We need to be careful since formulas and proofs both can be uniform or nonuniform. See figures 1.3 and 1.4 for possible combinations.

In the other direction, assume that theory $\mathcal{T}$ proves the soundness of a class of nonuniform proofs $\mathcal{F}$. If the nonuniform formulas and proofs can be generated uni-

		Uniform Tautology: $\forall X\ \varphi(X)$
Uniform Proof:	π	π: $\mathcal{T} \vdash \forall X\ \varphi(X)$

		Sectioned Tautology: $\{\forall X = n\ \varphi(X)\}_n$
Uniform Proof:	π	$\{\pi$: $\mathcal{T} \vdash \forall X = n\ \varphi(X)\}_n$
Sectioned Proof:	$\{\pi(n)\}_n$	$\{\pi(n)$: $\mathcal{T} \vdash \forall X = n\ \varphi(X)\}_n$
Proof with Advice:	$\{\pi(n, G(n))\}_n$	$\{\pi(n, G(n))$: $\mathcal{T} \vdash \forall X = n\ \varphi(X)\}_n$
Nonuniform Proof:	$\{\pi_n\}_n$	$\{\pi_n$: $\mathcal{T} \vdash \forall X = n\ \varphi(X)\}_n$

		Tautology with Advice: $\{\forall X = n\ \varphi(X, F(n))\}_n$
Uniform Proof:	π	$\{\pi$: $\mathcal{T} \vdash \forall X = n\ \varphi(X, F(n))\}_n$
Sectioned Proof:	$\{\pi(n)\}_n$	$\{\pi(n)$: $\mathcal{T} \vdash \forall X = n\ \varphi(X, F(n))\}_n$
Proof with Advice:	$\{\pi(n, G(n))\}_n$	$\{\pi(n, G(n))$: $\mathcal{T} \vdash \forall X = n\ \varphi(X, F(n))\}_n$
Nonuniform Proof:	$\{\pi_n\}_n$	$\{\pi_n$: $\mathcal{T} \vdash \forall X = n\ \varphi(X, F(n))\}_n$

		Nonuniform Tautology: $\{\forall X = n\ \varphi_n(X)\}_n$
Uniform Proof:	π	$\{\pi$: $\mathcal{T} \vdash \forall X = n\ \varphi_n(X)\}_n$
Sectioned Proof:	$\{\pi(n)\}_n$	$\{\pi(n)$: $\mathcal{T} \vdash \forall X = n\ \varphi_n(X)\}_n$
Proof with Advice:	$\{\pi(n, G(n))\}_n$	$\{\pi(n, G(n))$: $\mathcal{T} \vdash \forall X = n\ \varphi_n(X)\}_n$
Nonuniform Proof:	$\{\pi_n\}_n$	$\{\pi_n$: $\mathcal{T} \vdash \forall X = n\ \varphi_n(X)\}_n$

F and G are advice functions from natural numbers to binary strings.

Figure 1.4: Uniform and Nonuniform Proofs

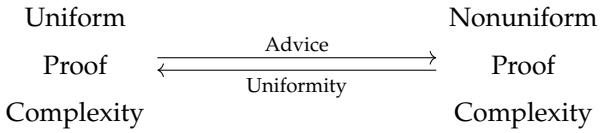


Figure 1.5: Uniformity and Advice for Proof Complexity

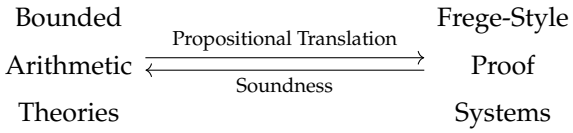


Figure 1.6: Uniform and Nonuniform Proof Models

formly inside the theory and the theory uniformly proves the correctness of the generated proofs, we can obtain a uniform proof. See figures 1.5 and 1.6.

We have already mentioned that the bounded arithmetic theory for a complexity class is obtained by adding a comprehension axiom. Moreover, the class of provably total functions of the theory turns out to be exactly those computable in the complexity class. Similarly, there are witnessing theorems in the reverse direction for nonuniform proof complexity classes which are obtained by restricting the cuts to circuits from the nonuniform complexity class[CN10]. Figure 1.7 states the connections for the complexity classes AC^0, NC^1, and P.

1.2 Motivation for the Thesis

There are two underlying themes behind our work. First, we are interested in transferring tools and methods from computational complexity to proof complexity. Consider lower bound results in nonuniform computational complexity. We know from [Ajt83; FSS84; Hås87] that parity requires exponential-size bounded-depth circuits. [Ajt88; BIK+92; Pit92; PBI93; KPW95] proved a corresponding result for bounded-depth Frege proofs: the pigeonhole principle tautologies require exponential-size bounded-depth Frege proofs. The result is obtained by proving a proof complexity version of Håstad's switching lemma for bounded-depth proofs. See [UF96] for a simplified exposition. Here the pigeonhole principle tautologies correspond to the parity function. Transferring results from computational complexity to proof complexity is not a straightforward task. E.g. it is known for almost 30 years [Raz87; Smo87] that bounded-depth

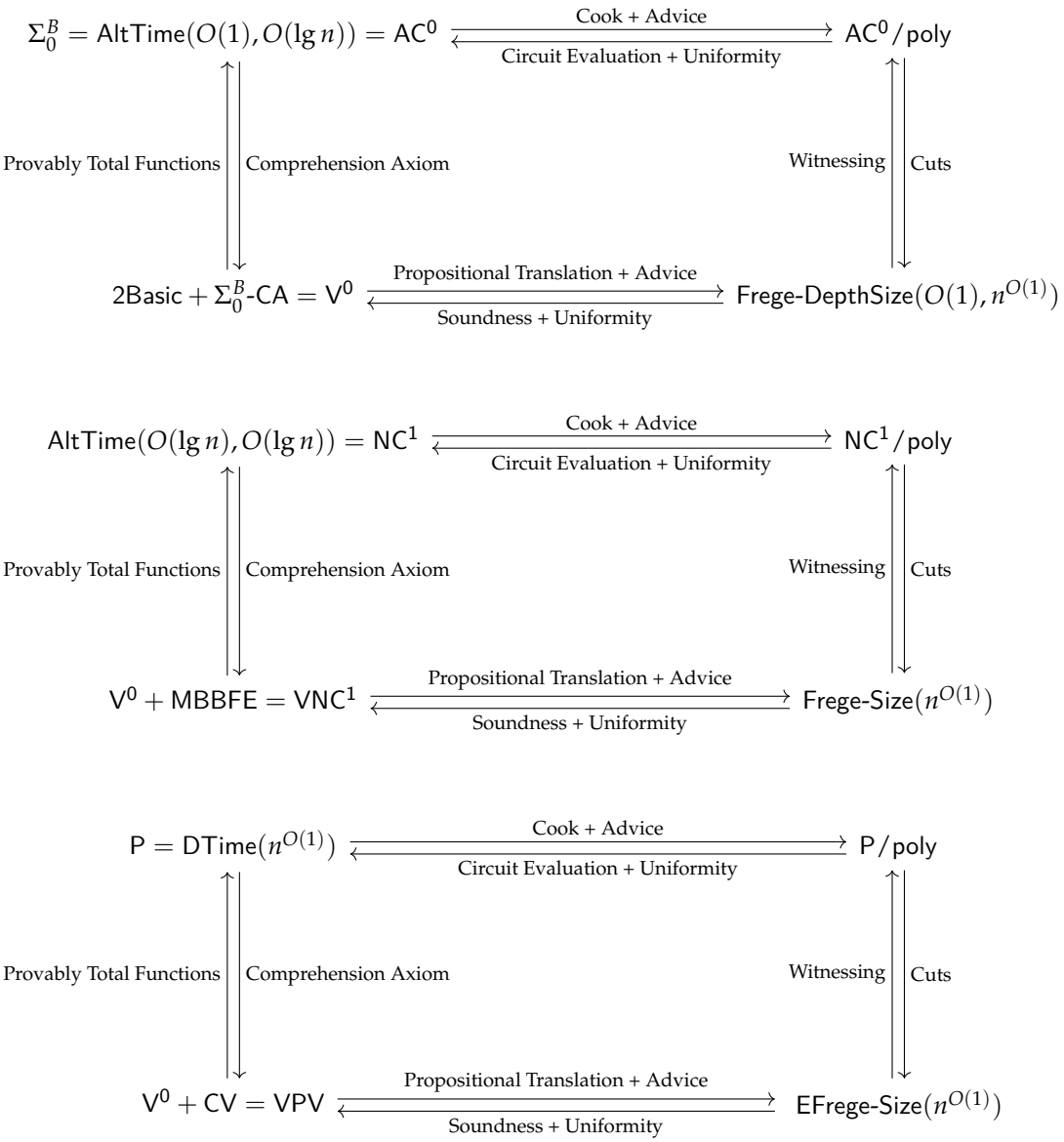


Figure 1.7: Diagram of Correspondences for AC^0, NC^1, and P

circuits with $\mod 2$ gates require exponential size to compute the $\mod 3$ function. However, despite considerable effort, a similar result for bounded-depth Frege proofs with formulas containing $\mod 2$ gates remains elusive. The idea of using tools from computational complexity theory in proof complexity can be achieved also by recasting questions from proof complexity as pure complexity theory questions. This idea has been developed in a systematic manner in [Kra10]. We still think transferring tools from computational complexity to proof complexity can help us to achieve a better understanding of proof complexity. Another example of this kind of connection is the relation between the PH hierarchy in computational complexity and Buss's S_2 hierarchy in bounded arithmetic [Bus86]. Yet another example is the connection between "natural proofs" in complexity theory and unprovability of circuit lower bounds in bounded arithmetic theories [RR94; Raz95b]. We are interested in uniform proof complexity results because it seems plausible that we may be able to exploit uniformity, e.g. obtain finer hierarchy results, and prove new separations in uniform proof complexity where a nonuniform version is not known similar to the separation of PP from the uniform class TC^0 in computational complexity [All99].

Inside this theme our research was motivated by the result from [FPS15] that subexponential-size bounded-depth Frege proofs simulate Frege proofs which provides an alternative proof of non-automatizability of Frege under plausible cryptographic conjectures [BPR00; BDG+04]. Our goal was to obtain a uniform version of this simulation. A uniform version would clarify the underlying concepts and essential requirements for the result and would make it possible to extend it to new stronger proof systems corresponding to other computational complexity classes like NL and SC. This would allow us to use strong lower bounds against bounded-depth Frege proofs to prove lower bounds against these stronger proof classes. Moreover, combining these results with uniformity may lead to stronger lower bounds for the uniform proof classes. In addition, the techniques developed may help towards proving a proof complexity version of [AK10].

We were interested in obtaining a proof complexity version of [AK10] which shows that if the Boolean formula evaluation problem (BFE) is inside TC^0 then it has circuits of size $n^{1+\varepsilon}$ for arbitrary small $\varepsilon > 0$. The contrapositive of this statement implies that to separate TC^0 from NC^1, a difficult open problem, we only need a slightly superlinear lower bound on the size of TC^0 circuits for BFE. A corresponding proof complexity statement would replace TC^0, NC^1, and BFE with TC^0-Frege, Frege, and soundness tautology for Frege.

The second underlying theme is clarifying and improving our understanding of the

role played by uniformity and nonuniformity in computational and proof complexity.

1.3 Contributions of the Thesis

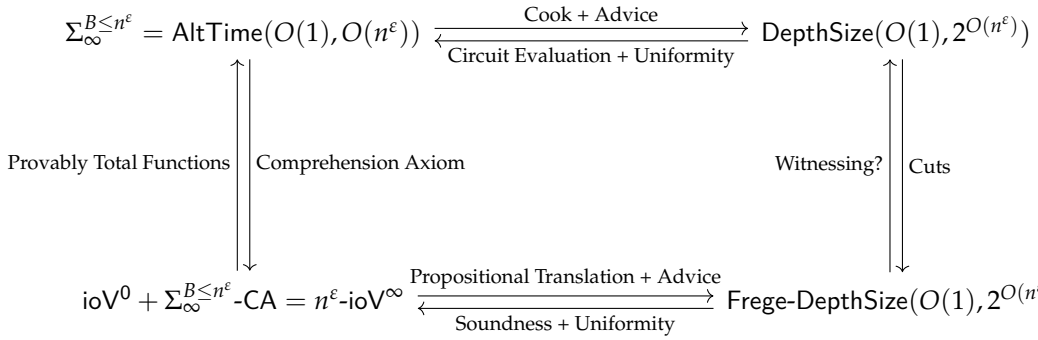
On the conceptual side, we clarify some notions and connections in proof complexity. The objective is to develop proof complexity in a more organized and systematic manner. For example, sometimes in the literature bounded-depth Frege is confusingly called a proof system despite the fact that it is *not* really a proof system but a *family* of proof systems. Polynomial-size bounded-depth Frege is a proof complexity class. We define the notion of proof complexity class, which allows us to properly discuss polynomial-size bounded-depth Frege and other interesting classes like subexponential-size bounded-depth Frege. The notion of computational complexity class plays a central role in computational complexity; the notion of proof complexity class is its proof complexity sister. This also allows us to discuss the notion of the universality of a class of formulas for a proof complexity class. Circuit evaluation problems and soundness statements take an even more central role. Figure 1.7 demonstrates some benefits of these conceptual clarifications.

Many interesting complexity classes like NP, polynomial-size DNF formulas, and SubExp are not closed under composition. However, the classes of functions associated with arithmetic theories are closed under composition. This is a major obstacle for designing bounded arithmetic theories for these classes. We provide a novel framework of io-types to develop theories for complexity classes which are not closed under composition. We use this framework to design a class of theories n^{ε}-ioV$^{\infty}$ for complexity classes $\mathsf{AltTime}(O(1), n^{\varepsilon})$ with $\varepsilon < 1$, which correspond to uniform succinct subexponential-size bounded-depth circuits.

We design a new proof system H for relativized quantified propositional formulas with Boolean function symbols. We provide a propositional translation from our theories to subsystems of H and transform the resulting proofs to bounded-depth Frege proofs. Our translation framework is systematic and clean and can be easily adapted to other axioms. It essentially reduces the problem of propositional translation to finding suitable explicit witnessing formulas for axioms and providing propositional proofs for the translation of axioms. See figure 1.8.

We prove and formalize the following containment between computational complexity classes in our n^{ε}-ioV$^{\infty}$ theories:

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



The relations between uniform and nonuniform computational and proof complexity classes:

- Uniform computation complexity class: $\mathsf{AltTime}(O(1), O(n^\varepsilon))$.
- Nonuniform computational complexity class: $\mathsf{DepthSize}(O(1), \exp(O(n^\varepsilon)))$.
- Uniform proof complexity class: $n^\varepsilon\text{-ioV}^\infty$.
- Nonuniform proof complexity class: $\mathsf{Frege\text{-}DepthSize}(O(1), \exp(O(n^\varepsilon)))$.
- An $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ algorithm can be transformed into a $\mathsf{DepthSize}(O(1), \exp(O(n^\varepsilon)))$ circuit family using Cook's translation from Turing machines to circuits (chapter 7). This result also holds when $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ algorithms are supplied with advice (advice is provided as an oracle).
- The evaluation problem for $\mathsf{DepthSize}(O(1), \exp(O(n^\varepsilon)))$ where circuits are given as oracles is in $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ (chapter 7, theorem 17). A uniform $\mathsf{DepthSize}(O(1), \exp(O(n^\varepsilon)))$ circuit whose connection language is in $\mathsf{DLogTime}$ can be transformed into an $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ algorithm.
- $n^\varepsilon\text{-ioV}^\infty$ is obtain by adding a comprehension axioms for $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ functions (section 8.1, definition 63).
- The class of provably total functions of $n^\varepsilon\text{-ioV}^\infty$ is $\mathsf{AltTime}(O(1), O(n^\varepsilon))$ (section 8.1, theorem 19).
- A Σ_0^B formula can be translated into a polynomial-size bounded-depth propositional formula family. An $n^\varepsilon\text{-ioV}^\infty$ proof of a Σ_0^B formula can be translated into a $\mathsf{Frege\text{-}DepthSize}(O(1), \exp(O(n^\varepsilon)))$ proof family of the translation of the Σ_0^B formula (section 8.1.2, theorem 20). The result also holds in the presence of additional function symbols and axioms.
- The soundness of $\mathsf{Frege\text{-}DepthSize}(O(1), \exp(O(n^\varepsilon)))$ proofs is provable in $n^\varepsilon\text{-ioV}^\infty$. The soundness of polynomial-size $n^\varepsilon\text{-bdG}_\infty$ proofs is also provably in $n^\varepsilon\text{-ioV}^\infty$ (section 8.1.3, theorem 21). A uniform polynomial-size $n^\varepsilon\text{-bdG}_\infty$ proof of the translation of a Σ_0^B formula can be transformed into a $n^\varepsilon\text{-ioV}^\infty$ proof of the Σ_0^B formula.

Figure 1.8: Diagram of Correspondences for $\mathsf{AltTime}(O(1), O(n^\varepsilon))$

Table 1.1: Our Uniform Proof Complexity Result

	Nonuniform	Uniform
Computational Complexity	$\mathsf{NC}^1/\mathsf{poly} \subseteq \mathsf{DepthSize}(O(1/\varepsilon), 2^{O(n^\varepsilon)})$ (Folklore)	$\mathsf{NC}^1 \subseteq \mathsf{AltTime}(O(1), O(n^\varepsilon))$ [AK10], (theorem 17)
Proof Complexity	$\mathsf{FregeSize}(n^{O(1)}) \subseteq O(1/\varepsilon)\text{-}\mathsf{FregeSize}(2^{O(n^\varepsilon)})$ [FPS15]	$\mathsf{ioVNC}^1 \subseteq n^\varepsilon\text{-}\mathsf{ioV}^\infty$ (theorem 26)

where $\delta > 0$ is arbitrary. We use the formalization to show that our n^ε-ioV$^\infty$ theories contain the bounded arithmetic theory ioVNC^1 corresponding to complexity class NC^1. This gives a uniform version of [FPS15]. See table 1.1.

We use the fact that soundness tautologies for Frege are universal for polynomial-size Frege proofs and are provable in ioVNC^1. This allows us to transform polynomial-size Frege proofs to subexponential-size bounded-depth Frege proofs.

We are aware that the correspondence between uniform and nonuniform computational complexity and proof complexity is not perfect. The general notion of a computational complexity class as an arbitrary set of computational tasks is too unstructured to be useful. Even when the class has a nice and robust structure like P, BPP, and P/poly, the correspondence is not one to one: the nonuniform versions of both P and BPP with a polynomial amount of advice turn out to be P/poly. The situation in proof complexity is not different. We need to clarify what we expect from a reasonable correspondence. In section 9.1 we express our view that a category theoretical perspective may shed some light on the situation if we can find nice subcategories of uniform and nonuniform proof classes with nice adjoint functors between them.

Regarding [AK10], we were unable to obtain a proof complexity version. See question 6 and the following discussion in chapter 9 on issues involved.

Many of the results in this thesis originally appeared in [GC13].

1.4 Related Work

A major part of our work was motivated by the goal of obtaining a uniform version of [FPS15]. We also reprove the main result from the uniform version. [Mül13] independently reproved the main result of [FPS15] through model-theoretic means by interpreting VNC^1 in polylogarithmic cuts of models of V^0 and formalizing Nepomnjascij's Theorem. Our results are proof-theoretic. In addition, we provide a theory for $\mathsf{AltTime}(O(1), n^\varepsilon)$ and use it to provide a uniform version. [Mül13] does not provide a uniform version. See table 1.1. Our method also applies to stronger theories like VNL since we can prove and formalize a result stronger than Nepomnjascij's theorem. The

stronger containment means that theories corresponding to complexity classes NL and SC are contained in our theory for $\mathsf{AltTime}(O(1), n^{\varepsilon})$. Therefore, proof systems for them can be converted to subexponential-size bounded-depth Frege proofs.

Our proof system H for relativized quantified propositional formulas is an extension of quantified propositional proof system G of [KP90]. A similar proof system called QPC(R) is introduced in [Coo12]. The proof system G and its subsystems are extensively studied [Kra95; CS99; CM05; Per09; CN10]. A nice property of our proof system is that it allows translated proofs to maintain the structure of proofs in bounded arithmetic theories.

The notion of "safe" and "normal" variables is introduced in [BC92] in the context of polynomial-time computable functions to control recursion and capture polynomial-time computable functions without explicit bounds. Their goal was to design a function algebra for FP. On the hindsight, it is possible to think of their algebra as an equational arithmetic theory with safe and normal types. In this way, we can view input and output types to correspond to normal and safe types. Alternatively, we can build an equational theory from our io-typed theories and consider the resulting typed function algebra. Another older related work is [CT86] where a bounded arithmetic theory is designed for the k-fold exponential functions, k-ExpF, which is not closed under composition. They achieve this by using k sorts and having an exponential function from each sort to the next one. This is similar to our idea of control composition using types. The issue of controlling the recursion and growth of functions comes up in various topics, most recently in implicit complexity theory and substructural logics [Dal12].

1.5 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 provides some preliminary results we need from computational and proof complexity from the literature. Chapter 3 introduces our io-typed bounded arithmetic theories. In chapter 4 we define our relativized quantified propositional proof system H and its subsystems. The chapter also introduces proof complexity classes and reductions between them and discusses the universality of soundness statements. Chapter 5 is devoted to propositional translation from io-typed theories to proof families in H. This provides the connection from bounded arithmetic theories to proof complexity classes. Chapter 6 is devoted to the reverse direction and its essential ingredient: the provability of soundness of proof complexity classes in theories. Chapter 7 provides the proof for

$\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)}) \subseteq \mathsf{AltTime}(O(1/\delta), O(n^{\delta}))$. In chapter 8 we introduce our theories n^{ε}-ioV^{∞} and ioVNC^1 and show that the former contains the latter. Combining it with the results from previous chapters we obtain an alternative proof of the result from [FPS15]. Chapter 9 summarizes our contributions and discusses some open problems for the future directions.

Chapter 2

Preliminaries

Notation 1. *We use "$\mathcal{X}$" to refer to some fixed expression of a mathematical statement $\mathcal{X}$ inside a language. E.g. "$\tau \vDash \varphi$" is a formula expressing that the truth assignment represented by propositional variables denoted by τ satisfies the formula represented by propositional variables denoted by φ. We use $\ulcorner \mathcal{X} \urcorner$ where $\mathcal{X}$ is a mathematical object for some fixed encoding of $\mathcal{X}$ by truth values $\top$ and $\bot$. E.g. $\ulcorner \pi \urcorner$ is the binary encoding of the proof π.*

Definition 1. *The circuit evaluation problem for the circuit class* $\mathsf{DepthSize}(d(n), s(n))$ *is the problem of computing the output of a given circuit in* $\mathsf{DepthSize}(d(n), s(n))$ *on a given input, with the promise that the input is a circuit from the class. If $\mathcal{C}$ is defined as the union of* DepthSize *classes, then the circuit evaluation problem for $\mathcal{C}$ is the set of circuit evaluation problems for its members. We use* $\mathcal{C}$-Eval *to denote the circuit evaluation problem for $\mathcal{C}$.*

Example 1. $\mathsf{DepthSize}(d, n^k)$-Eval *is the problem of computing the output of a circuit of size n^k and depth d on an input of size n.* AC^0-Eval *is the set of circuit evaluation problems, one problem for each depth and each size.*

Remark 1. *Note that we can reduce* $\mathsf{DepthSize}(d, n^k)$-Eval *to* $\mathsf{DepthSize}(d, n)$-Eval *by padding the input. In other words, the class of polynomial-size depth d circuits has a complete problem under a rather weak class of reductions. Therefore, the restriction on the size of the circuit is not required and we can simply consider the problem of evaluating a given depth d circuit on a given input. However, the restriction on depth is necessary. Otherwise, we would be evaluating polynomial circuits of arbitrary depth which is outside* AC^0*. Moreover,* AC^0 *does not have a complete problem with respect to a class of fix-depth reductions.*

Theorem 1. $\mathsf{DepthSize}(O(1), 2^{O(t(n))})$-Eval *are in* $\mathsf{AltTime}(O(1), O(t(n)))$.

Proof. This follows from the fact that the AC^0 circuit evaluation problems are in $\mathsf{DLogTime\text{-}uniformAC}^0 = \mathsf{AltTime}(\mathsf{O}(1), \mathsf{O}(\mathsf{t}(\mathsf{n})))$ by a padding argument. □

Definition 2 (Projections). *A function family $F = \{F_n\}_n$ is called a* projection *if each output bit of F depends on at most one bit of its input. If the size of $|F|$ is polynomially bounded, then it is called a* p-projection.

Example 2. *Every function whose output does not depend on its inputs is a projection.*

Example 3. *The identity function is a projection.*

Theorem 2. $\mathsf{DepthSize}(O(d(n)), O(t(n))\text{-}\mathsf{Eval}$ *is complete for* $\mathsf{AltTime}(O(d(n)), O(t(n)))$ *under first-order polynomial-projections.*

Proof. The proof is similar to [Imm99, Prop. 11.10]. □

Complexity classes like AC^0 and NC^1 are uniform using their original definition. Nonuniform versions are indicated as $\mathsf{AC}^0/\mathsf{poly}$ and $\mathsf{NC}^1/\mathsf{poly}$.

Formulas in the language of two-sorted bounded arithmetic are closely connected to formulas in descriptive complexity. E.g. FO formulas in descriptive complexity can be converted to Σ_0^B formulas of two-sorted bounded arithmetic and vice versa: the string length function $|\ |$ of bounded arithmetic acts like max in descriptive complexity; quantification over numbers with bound $\leq n^k$ is equivalent to a block of k quantifiers in descriptive complexity; quantification over strings of size n^k is equivalent to quantification over a k-ary relations in descriptive complexity; order corresponds to order in descriptive complexity; string membership $x \in Y$ with string Y of length n^k corresponds to a relation $R^Y(i_0^x, \ldots, i_{k-1}^x)$ in descriptive complexity where R^Y is the relation corresponding to Y and $x = \sum_{j<k} i_j^x n^j$.

A search problem for a relation $\mathcal{Q}$ is the task of finding a π such that $\mathcal{Q}(\pi, \varphi)$ for a given $\varphi \in \mathsf{dom}(\mathcal{Q})$. In multivalued function notation, given $\varphi \in \mathsf{dom}(\mathcal{Q})$ return a $\pi \in \mathcal{Q}(\varphi)$.

A many-one reduction from a search problem $\mathcal{Q}$ to another search problem $\mathcal{Q}'$ is a pair of functions R_i and R_o such that

$$\forall \varphi \in \mathsf{dom}(\mathcal{Q}) \ \left[\mathcal{Q}'(R_i(\varphi)) \neq \emptyset \wedge \forall \pi \in \mathcal{Q}'(R_i(\varphi)) \ R_o(\varphi, \pi) \in \mathcal{Q}(\varphi)\right].$$

A reduction is polynomially bounded if R_i and R_o are polynomially bounded functions. A reduction is polynomial time if R_i and R_o are computable in polynomial time.

Chapter 3

Uniform Proof Complexity: Bounded Arithmetic Theories with Input/Output Types

This chapter is devoted to setting up the framework for our first-order bounded arithmetic theories. We define *io-typed* theories (section 3.1) and use them to provide theories for complexity classes which are not closed under composition like bounded-depth subexponential-size circuits. Our base theory will be ioV^0 (section 3.3). The usual framework for defining bounded arithmetic theories cannot deal with such classes because the set of provably total functions of theories in the usual framework will be closed under composition.

We start by defining an io-typed version of LK and the base theory 2Basic for two-sorted bounded arithmetic in [CN10].

3.1 io-Typed Two-Sorted Bounded Arithmetics

We work with two-sorted theories with number and string types. The main idea of io-typed theories is to restrict composition by distinguishing between input objects and output objects. The intuition is that the input-type terms are going to be *small* (of linear size) while the output-type terms can be *large* (of polynomial size, like the original two-sorted theories of bounded arithmetic). We also think of strings as finite sets of numbers with explicit upper bounds on their members (the size of the string). A number is in a string if the corresponding bit in the string is 1. A *unary string* is a string with a single 1 bit. In other words, a unary string contains a single number. We can

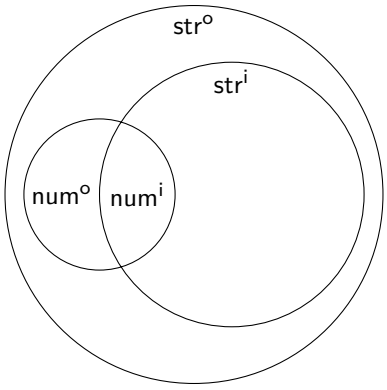


Figure 3.1: Venn Diagram of Sorts and Types

think of numbers as unary strings. Our semantics differs slightly from [CN10] 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, and the most significant bit does not need to be 1. E.g. 00000100 represents a binary string of length 8 containing only number 2. Since it only has a single 1 bit it also represents the number 2. The string 0101 represents a binary string of length 4 containing numbers 0 and 2. It is not a unary number because it has several 1 bits. See figure 3.1.

Remark 2. *There is a bijection between our models and the models of the language of* 2Basic *in [CN10]: Simply add a 1 to left side of each binary string. Because of this bijection, we can view strings as binary numbers.*

Remark 3. *Note that although we can think of numbers as a subset of strings there is nothing in the language $\mathcal{L}_2$ or our theories requiring that.*

Notation 2. *Lowercase letters denote numbers: a, b, c denote input-type numbers, and x, y, z denote output-type numbers. Uppercase 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; F, G for string-valued functions; s, t for number-valued terms; S, T for string-valued terms; φ, ψ for formulas; $\Gamma, \Delta, \Sigma, \Pi$ for formula sets; $\mathcal{S}$ for sequents; and $\mathcal{T}$ for sets of sequents, e.g. theories.*

Definition 3 (Language of io2Basic)**.** *The language $\mathcal{L}_2$ has two* sorts*:* num *for* (unary) numbers*, and* str *for* (binary) strings*; and two* types*:* i *for* input type*, and* o *for* output type*. Each term in the language has a sort and a type. The input types are subtypes of the output types, i.e. every object of input type is also an object of output type:* $\mathsf{num}^{\mathsf{i}} \subseteq \mathsf{num}^{\mathsf{o}} = \mathsf{num}$, $\mathsf{str}^{\mathsf{i}} \subseteq \mathsf{str}^{\mathsf{o}} = \mathsf{str}$.

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).*

The 0, 1, *and the function symbols* $+$, $\cdot$, pd *act only on numbers and produce numbers. Strings can only appear in* $\in$ *and* $|\ |$. *The length function* $|\ |$ *acts on strings and returns the length as a number. The membership relation* $x \in Y$ *is true when* $x < |Y|$ *and the* x*th bit of the string* Y *is 1.* $x \in Y$ *is false when* $|Y| \leq x$.

Every number/string term in the language is an output-type term of the same sort. The input-type terms of the language are a subset of output-type terms. 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.

Notation 3. *We abbreviate* $x \leq y \wedge x \neq y$ *as* $x < y$.

Notation 4. *We abbreviate* $|Y| = x$ *and* $|Y| \leq x$ *by* $Y = x$ *and* $Y \leq x$. *A bounded string quantifier is a string quantifier with an explicitly given size e.g.* $\exists Y = x$, $\forall Y = x$.

Notation 5. *Equality for strings,* $X = Y$, *is an abbreviation for* $|X| = |Y| \wedge \forall x \leq |X|\ (x \in X \leftrightarrow x \in Y)$.

Remark 4. *Note that unlike [CN10] string equality is* not *part of the language. This simplifies the propositional translation.*

Notation 6. *We abbreviate* $\forall x < |Y|\ (x \in Y \rightarrow \varphi)$ *as* $\forall x \in Y\ \varphi$.

Definition 4 (Σ_0^B, Σ_i^B, Π_i^B, Σ_∞^B)**.** *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_i^b *and* Π_i^b. *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_0^B. *A formula is* Σ_0^B *if it does not have any string quantifiers and all number quantifiers in it are bounded by terms in the language. Bounded formulas with at most* i *alternations of string quantifiers comprise the union of the classes* Σ_i^B *and* Π_i^B. *We define* $\Sigma_\infty^B = \bigcup_i \Sigma_i^B$.

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.

Definition 5 (Standard model $\mathbb{N}_2$ for $\mathcal{L}_2$)**.** *An* $\mathcal{L}_2$*-structure has four sets for interpreting* num, num^i, str, *and* str^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* num^i *as* 0^*10^* *and* str *and* str^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. The operations and relations of* $\mathcal{L}_2$ *have the standard intended interpretations as explained above.*

This more complex encoding of unary numbers is required since otherwise we cannot consider nonconstant number-valued functions in nonuniform models of computation. We need to be able to represent different unary numbers 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.

Notation 7. *We use m and n for numbers and M and N for strings in the standard model.*

Definition 6 (Size)**.** *The size of a number is the number itself. The size of a string is its length as a number.*

Definition 7 (Linear term)**.** *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 by a linear term in the size of its inputs.*

We adopt the sequent calculus LK of [CN10] with quantifier introduction rules to respect the types. See tables 3.1 and 3.2. In the input-type quantifier introduction rules, 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 $\exists$R and $\forall$L rules, if the quantification variable is of input type then the target term must be also of input type. Similarly, in $\forall$R and $\exists$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 $\forall a\ \varphi[x/a] \Rightarrow \forall x\ \varphi$ and $\exists x\ \varphi \Rightarrow \exists a\ \varphi[x/a]$. The implications in the other direction are still provable as expected: input types are subsets of output types, so $\forall x\ \varphi \Rightarrow \forall a\ \varphi[x/a]$ and $\exists a\ \varphi[x/a] \Rightarrow \exists x\ \varphi$ are valid.

We write $\pi : \mathcal{T} \vdash \varphi$ for "π is an LK-proof of φ in the theory $\mathcal{T}$", and $\mathcal{T} \vdash \varphi$ for "φ has an LK-proof in the theory $\mathcal{T}$".

Definition 8 (Parameters of a proof)**.** *We refer to the last sequent in a proof as its* end-sequent. Parameters *of an* LK *proof are the free variables in its end-sequent.*

Example 4. *A proof of* $\Rightarrow \exists X = |A|\ \forall x < |A|\ (x \in X \leftrightarrow x \in A)$ *is given in figure 3.2. The proof has a single parameter which is the variable* A*.*

$$\frac{}{\bot \Rightarrow}\,\bot \qquad\qquad \frac{}{\Rightarrow \top}\,\top$$

$$\frac{}{\varphi \Rightarrow \varphi}\,\mathbf{Id}$$

$\dfrac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma, \neg\varphi \Rightarrow \Delta}\,\neg\mathbf{L}$	Not	$\dfrac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi}\,\neg\mathbf{R}$
$\dfrac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, \varphi \wedge \psi \Rightarrow \Delta}\,\wedge\mathbf{L}$	And	$\dfrac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi}\,\wedge\mathbf{R}$
$\dfrac{\Gamma, \varphi \Rightarrow \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \vee \psi \Rightarrow \Delta}\,\vee\mathbf{L}$	Or	$\dfrac{\Gamma \Rightarrow \Delta, \varphi, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi}\,\vee\mathbf{R}$
$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma, \varphi \Rightarrow \Delta}\,\mathbf{WL}$	Weakening	$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi}\,\mathbf{WR}$
$\dfrac{\Gamma, \varphi, \varphi \Rightarrow \Delta}{\Gamma, \varphi \Rightarrow \Delta}\,\mathbf{CL}$	Contraction	$\dfrac{\Gamma \Rightarrow \Delta, \varphi, \varphi}{\Gamma \Rightarrow \Delta, \varphi}\,\mathbf{CR}$
$\dfrac{\Gamma_1, \psi, \varphi, \Gamma_2 \Rightarrow \Delta}{\Gamma_1, \varphi, \psi, \Gamma_2 \Rightarrow \Delta}\,\mathbf{XL}$	Exchange	$\dfrac{\Gamma \Rightarrow \Delta_1, \psi, \varphi, \Delta_2}{\Gamma \Rightarrow \Delta_1, \varphi, \psi, \Delta_2}\,\mathbf{XR}$

$$\frac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\,\mathbf{Cut}$$

Table 3.1: PK

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \in A \Rightarrow x \in A}}{\Rightarrow \neg x \in A, x \in A}\,\neg\mathbf{R}}{\Rightarrow x \in A \to x \in A}\,\vee\mathbf{R} \qquad \dfrac{\dfrac{\dfrac{}{x \in A \Rightarrow x \in A}}{\Rightarrow \neg x \in A, x \in A}\,\neg\mathbf{R}}{\Rightarrow x \in A \to x \in A}\,\vee\mathbf{R}}{\Rightarrow x \in A \leftrightarrow x \in A}\,\wedge\mathbf{R}}{\Rightarrow \neg x < |A|, x \in A \leftrightarrow x \in A}\,\mathbf{WR}}{\Rightarrow x < |A| \to (x \in A \leftrightarrow x \in A)}\,\vee\mathbf{R}}{\Rightarrow \forall x < |A|\ (x \in A \leftrightarrow x \in A)}\,\forall\mathbf{R}$$

$$\dfrac{\dfrac{}{\Rightarrow |A| = |A|} \qquad \Rightarrow \forall x < |A|\ (x \in A \leftrightarrow x \in A)}{\Rightarrow |A| = |A| \wedge \forall x < |A|\ (x \in A \leftrightarrow x \in A)}\,\wedge\mathbf{R}$$

$$\dfrac{\Rightarrow |A| = |A| \wedge \forall x < |A|\ (x \in A \leftrightarrow x \in A)}{\Rightarrow \exists X = |A|\ \forall x < |A|\ (x \in X \leftrightarrow x \in A)}\,\exists\mathbf{R}$$

Figure 3.2: LK Proof of $\Rightarrow \exists X = |A|\ \forall x < |A|\ (x \in X \leftrightarrow x \in A)$

Existential Quantifier Rules

$$\frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \exists x\ \varphi \Rightarrow \Delta}\ \exists\mathbf{L} \qquad \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \exists a\ \varphi \Rightarrow \Delta}\ \exists\mathbf{L} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi[x/t]}{\Gamma \Rightarrow \Delta, \exists x\ \varphi}\ \exists\mathbf{R} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi[a/t]}{\Gamma \Rightarrow \Delta, \exists a\ \varphi}\ \exists\mathbf{R}$$

$$\frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \exists X\ \varphi \Rightarrow \Delta}\ \exists\mathbf{L} \qquad \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \exists A\ \varphi \Rightarrow \Delta}\ \exists\mathbf{L} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi[X/T]}{\Gamma \Rightarrow \Delta, \exists X\ \varphi}\ \exists\mathbf{R} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi[A/T]}{\Gamma \Rightarrow \Delta, \exists A\ \varphi}\ \exists\mathbf{R}$$

Universal Quantifier Rules

$$\frac{\Gamma, \varphi[x/t] \Rightarrow \Delta}{\Gamma, \forall x\ \varphi \Rightarrow \Delta}\ \forall\mathbf{L} \qquad \frac{\Gamma, \varphi[a/t] \Rightarrow \Delta}{\Gamma, \forall a\ \varphi \Rightarrow \Delta}\ \forall\mathbf{L} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \forall x\ \varphi}\ \forall\mathbf{R} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \forall a\ \varphi}\ \forall\mathbf{R}$$

$$\frac{\Gamma, \varphi[X/T] \Rightarrow \Delta}{\Gamma, \forall X\ \varphi \Rightarrow \Delta}\ \forall\mathbf{L} \qquad \frac{\Gamma, \varphi[A/T] \Rightarrow \Delta}{\Gamma, \forall A\ \varphi \Rightarrow \Delta}\ \forall\mathbf{L} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \forall X\ \varphi}\ \forall\mathbf{R} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \forall A\ \varphi}\ \forall\mathbf{R}$$

Restrictions on variables:

- x, X, a, A in $\exists$L and $\forall$R are called *eigenvariables* and cannot appear freely in their bottom sequent.
- t and T in $\forall$L and $\exists$R are called *target terms* and cannot have bound variables.

Restrictions on types:

- In the input-type quantifier rules, the target term must be of input-type.
- In the output-type quantifier rules, the eigenvariable must be of output-type.

Table 3.2: LK

Definition 9 (Definability in the standard model)**.** *Let Φ be a set of formulas (for example take $\Phi = \exists^B\Sigma_0^B$). We say that a relation P in the standard model is Φ-definable iff there is a formula $\varphi \in \Phi$ with free variables $\vec{x}$ and $\vec{X}$ such that $P = \{(\vec{n}, \vec{N}) \in \mathbb{N}_2 \mid \mathbb{N}_2 \vDash \varphi[\vec{x}, \vec{X}/\vec{n}, \vec{N}]\}$. The graph of a function f in the standard model is defined as $\{(\vec{n}, \vec{N}, m) \in \mathbb{N}_2 \mid f(\vec{n}, \vec{N}) = m\}$.*

Remark 5 (Σ_0^B and AC^0)**.** *The Σ_0^B definable relations comprise functions computable in $\mathsf{AltTime}(O(1), O(\lg n))$ — which are the same as functions computable by AC^0 circuits. See [CN10, Theorem IV.3.6].*

Definition 10 (Definability in a theory)**.** *Let f be a number-valued function. We say that a function f is Φ-definable in $\mathcal{T}$ iff its graph is Φ-definable using a formula $\varphi(\vec{x}, \vec{X}, y) \in \Phi$ and $\mathcal{T}$ proves that φ defines a function, i.e. $\mathcal{T} \vdash \exists! y \leq t(\vec{x}, \vec{X})\ \ \varphi(\vec{x}, \vec{X}, y)$ for some term $t(\vec{x}, \vec{X})$.*

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 $\mathcal{T}$ iff there is a formula $\varphi(\vec{x}, \vec{X}, z) \in \Phi$ and a number-valued term $t(\vec{x}, \vec{X})$ such that

- *t defines the length of F over the standard model, i.e. $\mathbb{N}_2 \vDash |F(\vec{n}, \vec{N})| = t(\vec{n}, \vec{N})$,*
- *φ defines the bit-graph of the function F over the standard model, and*
- *$\mathcal{T}$ proves that φ and t define a total function, i.e.*

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

Remark 6. *Note that in circuit complexity, the size of the output of a function must only depend on the size of its inputs. Therefore, we require the size of F depends only on the size of its arguments.*

Definition 11 (io-typed provably total functions)**.** *We say that a function is* provably total *in a theory $\mathcal{T}$ if the function is Φ-definable in $\mathcal{T}$. The class of* io-typed provably total functions *of a theory $\mathcal{T}$ consists of those functions that $\mathcal{T}$ 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.*

Remark 7. *Every provably total function is also io-typed provably total. However, the converse need not be true. For a function to be io-typed provably total it is sufficient for the function to be total over input-types. Therefore, the class of io-typed provably total functions of a theory can be larger class than the usual class of its provably total functions.*

The class of io-typed provably total functions of a theory is the class of functions we associate with the theory.

Remark 8. *The appropriate class Φ of formulas in the definition above depends on the class we want to capture. We want the provably total functions in $\mathcal{T}$ to be those in the complexity class associated with $\mathcal{T}$. For two-sorted theories associated with complexity classes contained in polynomial time (such as* ioV^0 *defined below), the right choice is $\Phi = \exists^B\Sigma^B_0$ [CN10]. For the theory $t(n)$-ioV^∞ defined in section 8.1 we choose Φ to be a larger class capable of expressing functions computable in* $\mathsf{AltTime}(O(1), O(t(n)))$.

Definition 12 (Extension by definition)**.** *When a function is provably total in a theory $\mathcal{T}$, we can add a new function symbol to the language for it and include its definition as an axiom in the theory to obtain an* extension by definition *of $\mathcal{T}$.*

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.

Definition 13 (Substring)**.** *Consider the substring of X starting at bit y and of length z which we denote by $X[y, z]$. It can be defined as follows:*

- $|X[y, z]| := z$,
- $x \in X[y, z] := x < z \wedge y + x \in X$.

Example 5. *Let $X = 10110101$, $y = 5$, and $z = 3$. The substring $X[y, z]$ is the string* 11.

Definition 14 (Fractional power)**.** *Consider the* fractional power $\lfloor x^{\frac{1}{d}} \rfloor$, *which we will write simply as $x^{\frac{1}{d}}$, where d is a fixed positive integer. It can be defined as follows:*

- $x^{\frac{1}{d}} = y := y^d \leq x \wedge x < (y+1)^d$.

Remark 9. *The theory* ioV^0 *defined later in this chapter is powerful enough to prove that the fractional power function and substring are total and have linear growth. Therefore, we can add them to the language and consider the input terms are closed under them when dealing with theories containing* ioV^0.

Remark 10. *When we extend the language of a theory by definitions the language will contain new formulas. We have to check if the axiom schemes of the theory (e.g. comprehension) hold for the enlarged class of formulas. For cases of interest like the ones above the techniques presented in [CN10] suffice to show this is the case for our base theory* ioV^0. *We will assume that the axiom schemes for theories build upon our base theory are for formulas in the extended language.*

B1	$x+1 \neq 0$	B7	$x \leq y \wedge y \leq x \rightarrow x = y$
B2	$x+1 = y+1 \rightarrow x = y$	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	$\mathsf{pd}(0) = 0 \wedge (x \neq 0 \rightarrow \mathsf{pd}(x)+1 = x)$
L	$y \in X \rightarrow y < \lvert X \rvert$		

Table 3.3: io2Basic

3.2 Theory io2Basic

We start by defining an io-typed version of 2Basic of [CN10].

Definition 15 (io2Basic)**.** *The theory* io2Basic *is given by axioms in table 3.3.*

Remark 11. *Note that unlike the original* 2Basic *in [CN10], 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 [Bus86]. A binary string is determined by its length and its bits. This change does not make any essential difference in the presence of number induction for Σ^B_0 formulas: the original version of the length function is definable.*

3.3 Theory ioV^0 for AC^0

Our theory ioV^0 is an io-typed version of the base theory V^0 of [CN10] corresponding to the complexity class AC^0. Besides the axioms for io2Basic, we need an io-typed axiom for induction, an io-typed axiom scheme for comprehension, and two type-conversion axioms, one for each sort.

Definition 16 (The induction and comprehension axioms)**.** *The induction and comprehension axioms for io-typed theories are defined as follows:*

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

In the axiom φ-CA the formula φ can contain variables which are free in the axiom. However, all variables which are free in the axiom must be of input type.

Remark 12. *We could have used a stronger version of Σ^B_0-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.*

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.

Definition 17 (The conversion axioms)**.** *The conversion axioms for io-typed theories are defined as follows:*

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

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, i.e. $b = \min(x, a)$ 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.

There are stronger forms of conversion, e.g. a comprehension axiom from output types to input types. However, these variations do not affect our results and we take these weaker forms.

The theory ioV^0 is obtained from io2Basic by adding the comprehension axiom for Σ^B_0 formulas, the induction axiom, and the conversion axioms.

Definition 18. *The io-typed theory* ioV^0 *is defined as follows:*

- $\mathsf{ioV}^0 := \mathsf{io2Basic} + \mathsf{Ind} + \Sigma^B_0\text{-}\mathsf{CA} + \mathsf{oiConv}$.

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^B_0 formulas. Since ioV^0 has comprehension for these sets, polynomially-bounded functions with bit graphs in these sets are $\exists^B\Sigma^B_0$ definable functions in the theory. By a witnessing theorem, they are the only $\exists^B\Sigma^B_0$ definable functions in the theory. Thus the provably total functions in ioV^0 coincide with the AC^0 functions, where we take the class Φ associated with this theory to be $\exists^B\Sigma^B_0$. As a general rule, in our theories the comprehension axiom of the theory determines the computational power of the theory.

Theorem 3. *The class of provably total functions of* ioV^0 *is* AC^0.

Proof. We first prove the witnessing, i.e. every provably total function of ioV^0 is in AC^0. We provide both a model-theoretic argument and a proof-theoretic argument to illustrate the connection between io-theories with the two sorted theories of [CN10].

Let ψ be an arbitrary formula in the language of io2Basic. We can view ψ as a formula in the language of 2Basic by ignoring the distinction between input and output types. We show that $\mathsf{V}^0 \vdash \psi$. Let $\mathcal{M}$ be an arbitrary model of V^0. We can view $\mathcal{M}$ as a model of ioV^0 where $\mathsf{str}^{\mathsf{i}} = \mathsf{str}^{\mathsf{o}}$ and $\mathsf{num}^{\mathsf{i}} = \mathsf{num}^{\mathsf{o}}$. It is straightforward to check that all axioms of ioV^0 hold in this model and $\mathcal{M} \vDash \mathsf{ioV}^0$. Therefore, $\mathcal{M} \vDash \psi$. Since $\mathcal{M}$ was an arbitrary model of V^0 it follows that $\mathsf{V}^0 \vdash \psi$. Therefore, any provably total function of ioV^0 is in AC^0.

Proof-theoretically, we can view any proof in ioV^0 as a proof in V^0 by ignoring the distinction between input and output types.

We cannot directly rely on untyped theories of [CN10] for bootstrapping as io-typed theories can be weaker. Let F be an AC^0 function with a polynomial length $t(\vec{n}, \vec{N}) = |F(\vec{n}, \vec{N})|$. Since the bit-graph of any AC^0 function can be represented by a Σ^B_0 formula, there is a $\varphi(\vec{a}, \vec{A}, z) \in \Sigma^B_0$ that represents the bit-graph of F. We have to show that $\mathsf{ioV}^0 \vdash \exists! Y = t(\vec{a}, \vec{A})\ \forall z < t(\vec{a}, \vec{A})\ (z \in Y \leftrightarrow \varphi(\vec{a}, \vec{A}, z))$. The existence follows from φ-CA. The uniqueness follows from the fact that two strings are equal if their length and their bits are equal. □

3.4 Theory ioVC

We can use ioV^0 to define io-typed versions of other theories built upon V^0. However, simply adding the same comprehension axiom used for VC in [CN10] 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 ioVC 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 and AC^0

functions[1], i.e. $\mathsf{TC}^0 = \overline{\mathsf{MAJ} + \mathsf{AC}^0}$. This result is not useful for defining the io-typed version of the theory. Another example is the theory VNC^1, which 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 $\mathcal{C}$ are as follows:

- ioVC has enough comprehension to evaluate problems in $\mathcal{C}$.
- The provably total functions of ioVC are exactly the functions in $\mathcal{C}$.

We use this framework to design theories for subexponential-size bounded-depth Frege (section 8.1) and polynomial-size Frege (section 8.2) proof systems

[1]The function MAJ computes the majority for the rows of a given matrix, not a binary single string. The result follows from the descriptive complexity result that TC^0 is captured by language FOM, FO with majority quantifiers. This does not hold for the majority function unless TC^0 collapses.

Chapter 4

Nonuniform Proof Complexity: Classes and Reductions

This chapter is devoted to nonuniform proof complexity. In this chapter we introduce the notions of *proof complexity classes* (section 4.2) and *proof reductions* between them (section 4.3), and prove the *universality* of the *soundness* of proof classes with respect to suitable classes of proof reductions (theorem section 4.6). The soundness tautologies for proof complexity classes are analogous to circuit evaluation problems for circuit complexity classes.

The need for the notions of proof complexity classes and proof reductions arises from the typically ad hoc treatment of proof classes like polynomial-size bounded-depth Frege proofs — which is *not* a proof system — in the literature. We provide an analogy to nonuniform computational complexity theory using these notions: a proof system is akin to a model of computation; size and depth are complexity measures over proofs as they are over circuits; a formula family belongs to a nonuniform proof complexity class iff it has a proof family in that proof complexity class; nonuniform proof classes like polynomial-size bounded-depth Frege are counterparts to circuit complexity classes like AC^0, the class of polynomial-size bounded-depth circuits; and soundness formulas are universal for proof classes as circuit evaluation problems are complete for computational complexity classes.

We will focus on *relativized quantified propositional formulas* with propositional function symbols (section 4.1) and the relativized quantified propositional proof system H for them (section 4.2). The proof system H for relativized quantified propositional formulas is obtained by extending Krajíček and Pudlák's quantified propositional proof system G [KP90] with propositional function symbols and by adding an *extensionality rule* for those function symbols. A proof system similar to H called QPC(R)

is introduced in [Coo12]. Our extensionality rule Ext corresponds to the axiom AX of QPC(R).

4.1 Relativized Quantified Propositional Formulas

We first define the class of relativized quantified propositional formulas. The relativized quantified propositional formulas are the nonuniform counterparts to relativized Σ^B_∞ formulas representing functions in the relativized polynomial hierarchy.

Notation 8. *The letters p, q, and r range over propositional variables; α, β and γ range over propositional function symbols; φ and ψ range over formulas; Γ, Δ, Σ, and Π range over sets/sequences of formulas. Each function symbol α has an arity denoted by $|\alpha|$.*

Definition 19 (Relativized quantified propositional formula)**.** *The logical symbols consist of $\top$, $\bot$, $\neg$, $\vee$, $\wedge$, $\exists$, and $\forall$. We have an infinite set of propositional variables denoted by $\mathcal{V}$. Let $\mathcal{L}$ be the set of propositional function symbols. The set of* relativized quantified propositional formulas *of $\mathcal{L}$, denoted by $\mathsf{Form}_\mathcal{L}$, is defined inductively:*

- *Propositional constants: $\top, \bot \in \mathsf{Form}_\mathcal{L}$.*
- *Propositional variables: $\mathcal{V} \subseteq \mathsf{Form}_\mathcal{L}$.*
- *Propositional functions: If $\alpha \in \mathcal{L}$ and $\vec{\varphi} = \varphi_0, \ldots, \varphi_{|\alpha|-1} \in \mathsf{Form}_\mathcal{L}$ then $\alpha(\vec{\varphi}) \in \mathsf{Form}_\mathcal{L}$.*
- *Negation: If $\varphi \in \mathsf{Form}_\mathcal{L}$ then $(\neg\varphi) \in \mathsf{Form}_\mathcal{L}$*
- *Disjunction and conjunction: If $\varphi, \psi \in \mathsf{Form}_\mathcal{L}$ then $(\varphi \vee \psi), (\varphi \wedge \psi) \in \mathsf{Form}_\mathcal{L}$.*
- *Existential and universal propositional quantifiers: If $\varphi \in \mathsf{Form}_\mathcal{L}$ and $p \in \mathcal{V}$ then $(\exists p\ \varphi), (\forall p\ \varphi) \in \mathsf{Form}_\mathcal{L}$*

Convention 1. *When writing formulas we may omit some parentheses to enhance readability as long as their omission does not cause ambiguity.*

Definition 20 ($\Sigma^q_i, \Pi^q_i, \Sigma^q_\infty$)**.** *The set of relativized quantified propositions formulas with i alternations of quantifiers starting with an existential (universal) quantifier are denoted by $\Sigma^q_i(\mathcal{L})$ (respectively $\Pi^q_i(\mathcal{L})$), the set of quantifier-free formulas by $\Sigma^q_0(\mathcal{L}) = \Pi^q_0(\mathcal{L})$, and the set of all quantified propositional formulas by $\Sigma^q_\infty(\mathcal{L}) = \Pi^q_\infty(\mathcal{L})$.*

Notation 9. *We use the following abbreviations:*

- *$\varphi \to \psi$ for $\neg\varphi \vee \psi$*

- $\varphi \leftrightarrow \psi$ *for* $(\varphi \to \psi) \wedge (\psi \to \varphi)$
- $\bigvee \vec{\varphi}$ *and* $\bigvee_{i<n} \varphi_i$ *for* $((\varphi_0 \vee \dots) \vee \varphi_{n-2}) \vee \varphi_{n-1}$ *where* $\vec{\varphi} = \varphi_0, \dots, \varphi_{n-1}$.
- $\bigwedge \vec{\varphi}$ *and* $\bigwedge_{i<n} \varphi_i$ *for* $((\varphi_0 \wedge \dots) \wedge \varphi_{n-2}) \wedge \varphi_{n-1}$ *where* $\vec{\varphi} = \varphi_0, \dots, \varphi_{n-1}$.
- $\exists \vec{p}\ \ \varphi$ *for* $\exists p_0 \dots \exists p_{n-1}\ \ \varphi$ *where* $\vec{p} = p_0, \dots, p_{n-1}$.
- $\forall \vec{p}\ \ \varphi$ *for* $\forall p_0 \dots \forall p_{n-1}\ \ \varphi$ *where* $\vec{p} = p_0, \dots, p_{n-1}$.
- $\vec{\varphi} \leftrightarrow \vec{\psi}$ *for* $\bigwedge_{i<n} (\varphi_i \leftrightarrow \psi_i)$ *where* $\vec{\varphi} = \varphi_0, \dots, \varphi_{n-1}$ *and* $\vec{\psi} = \psi_0, \dots, \psi_{n-1}$

Definition 21 (Sequent)**.** *A* sequent *consists of a pair of (finite) formula sequences. Let* $\mathcal{S}$ *be the sequent corresponding to the pair* (Γ, Δ)*. We denote* $\mathcal{S}$ *by the* $\Gamma \Rightarrow \Delta$*. The intended meaning of* $\Gamma \Rightarrow \Delta$ *is* $\bigvee_{\varphi \in \Gamma} \neg\varphi \vee \bigvee_{\psi \in \Delta} \psi$.

Definition 22 (Substitution)**.** *We say that a term is free in a formula iff all of its variables are free in the formula.* $\varphi[p/\psi]$ *denotes the formula resulting from* substituting ψ *for* p *in* φ*. The usual restrictions on substitution apply: only free occurrences are replaced, and the replacements must not become bound.*

Definition 23 (Semantics for formulas)**.** *A* truth assignment *is a function* $\tau : \mathcal{V} \to \{\top, \bot\}$*. A* model *for relativized quantified propositional formulas associates a propositional function with each function symbol in* $\mathcal{L}$*. We use* $\tau, \mathcal{M} \vDash \varphi$ *to express that the formula* φ *is* true *under the truth assignment* τ *and the model* $\mathcal{M}$*. When* φ *is true under all truth assignments and all models we write* $\vDash \varphi$ *and say that* φ *is* valid*. We refer to the set of valid formulas as* tautologies *and denote the set of tautologies by* $\mathsf{Taut}_{\mathcal{L}}$.

Convention 2. *We omit the model in case* $\mathcal{L}$ *is empty.*

Definition 24 (Semantics for sequents)**.** *A sequent* $\Gamma \Rightarrow \Delta$ *is true (under* τ *and* $\mathcal{M}$*) iff there is* $\varphi \in \Gamma$ *which is false or there is* $\psi \in \Delta$ *which is true. In other words,* $\Gamma \Rightarrow \Delta$ *if true iff* $\bigvee_{\varphi \in \Gamma} \neg\varphi \vee \bigvee_{\psi \in \Delta} \psi$ *is true.*

Remark 13. *The satisfiability problem for relativized quantified propositional formulas is complete for* NExpTime*. The validity is complete for* coNExpTime*. See [Coo12].*

Definition 25. *A* family of objects *is an infinite sequence of objects. We call a family of objects* an object family.

Example 6. *A* family of formulas *is an infinite sequence of formulas. Let* PHP_n *be the formula* $\bigwedge_{i\leq n+1}\bigvee_{k\leq n} p_{ik} \to \bigvee_{i\neq j\leq n+1}\bigvee_{k\leq n}(p_{ik}\wedge p_{jk})$*. Then the infinite sequence* $\{\mathsf{PHP}_n\}_n$ *is a family of formulas. We use* PHP *to denote the formula family* $\{\mathsf{PHP}_n\}_n$*.*

Convention 3. *We use the same symbols for objects as for families of objects. E.g. if* $\{\varphi_n\}_n$ *is a family of formulas, we use* φ *for the family* $\{\varphi_n\}_n$*.*

Next, we define a number of complexity measures on formulas. The formula tree of a formula is its construction tree where nodes are labeled with propositional variables, nonlogical symbols, logical symbols, or quantifiers. For a tree T with labeled nodes, and a set of labels L, the L-depth of T is the maximum number of labels from L in any branch of T. For $\bigwedge$ (also $\bigvee$, $\exists$, and $\forall$) we ignore the consecutive repetitions of $\wedge$ (respectively $\vee$, $\exists$, and $\forall$). The L-depth of a formula is the L-depth of its construction tree. We define the following complexity measures over quantified propositional formulas:

Definition 26 (Size, logical depth, quantifier depth)**.** *Let* φ *be a quantified propositional formula.*

- Size*:* $\mathsf{size}(\varphi)$ *is the number of symbols in* φ*.*
- Logical depth*:* $\mathsf{ldepth}(\varphi)$ *is the* $\{\neg,\bigvee,\bigwedge\}$*-depth of* φ*.*
- Quantifier depth*:* $\mathsf{qdepth}(\varphi)$ *is the* $\{\exists,\forall\}$*-depth of* φ*.*

We extend these to sets of formulas, sequences of formulas, sequents, etc. The size of a collection of objects is the total sum of the sizes of objects in it. The depth of a collection of objects is the max depth of the objects in it.

Let μ *be a complexity measure. The* μ *complexity of the family* $\{\varphi_n\}_n$ *is the function* $n\mapsto\mu(\varphi_n)$*.*

Definition 27 (Formula complexity class)**.** *Let* $\mathcal{C}$ *be a class of formulas, e.g. relativized quantified propositional formulas. A* $\mathcal{C}$ formula complexity class *is a set of* $\mathcal{C}$*-formula families.*

Typically a formula complexity class is defined by putting restrictions on the complexity of formulas w.r.t. complexity measures. We will study subclasses of relativized quantified propositional formulas with size, logical depth, and quantifier depth restrictions. We will consider the following bounds $O(1)$, n^{ε}, $n^{O(1)}$, $2^{O(n^{\varepsilon})}$ where $\varepsilon=\frac{1}{d}$.

Example 7. *The formula family* PHP *from example 6 belongs to the formula complexity class polynomial-size bounded-depth propositional formulas.*

Definition 28 (Concrete substitution instance)**.** *A* concrete instance *of a formula is obtained by substituting (some of) its free variables with $\top$ and $\bot$. We denote the set of concrete instances of a set of formulas Γ by $\overline{\Gamma}$.*

4.2 Relativized Quantified Propositional Proof System H

Definition 29 (Proof system)**.** *Let Q be a relation on two binary string inputs. We say π is a* Q-proof *for φ iff $Q(\pi, \varphi)$ accepts, in which case we write $\pi : Q \vdash \varphi$. We say φ is* provable *in Q iff φ has a Q-proof. A relation Q is a(n efficient)* proof system *for $\mathsf{Taut}_{\mathcal{L}}$ iff it satisfies the following conditions:*

- efficiency*: Q is computable in polynomial time, i.e. $Q \in \mathsf{P}$.*
- completeness*: every $\mathcal{L}$-tautology is provable in Q.*
- soundness*: every $\mathcal{L}$-formula provable in Q is an $\mathcal{L}$-tautology.*

We are interested in *families* of proofs for *families* of formulas.

Definition 30 (Proof family)**.** *We say a proof family $\{\pi_n\}_n$ is a Q-proof for a formula family $\{\varphi_n\}_n$ and write $\{\pi_n\}_n : Q \vdash \{\varphi_n\}_n$ iff for all n, $\pi_n : Q \vdash \varphi_n$.*

Definition 31 (Frege-style proof system)**.** *A* Frege-style *proof system Q is given by a finite set of rules. A* rule *is given by a* bottom sequent *and a finite list of* top sequents*. The list of top sequents can be empty, in which case we call it an* axiom*. An* instance *of a rule is a substitution instance of the rule obtained by substituting formulas for free propositional variables in it. A Q-proof for a sequent S is a finite labeled rooted dag where the labels for each node and its parents form an instance of a rule in Q, and the label of the root is S. We refer to S as the* end-sequent *of the proof. A proof for the formula φ is a proof for the sequent $\Rightarrow \varphi$.*

Definition 32 (Frege-style proof system with axioms)**.** *Let Γ be a set of formulas. We use Γ, Q to denote the proof system obtained from Q by adding the sequents in Γ as axioms. We write $\Gamma, Q \vdash S$ when there is a Q-proof for S using sequents in Γ as axioms.*

Next, we define some complexity measures on proofs.

Definition 33 (Size, logical depth, quantifier depth)**.** *Let Q be a Frege-style proof system. Let π be a Q-proof.*

- Size*: $\mathsf{size}(\pi) := \sum_{\varphi \in \pi} \mathsf{size}(\varphi)$.*

- Logical depth: $\mathsf{ldepth}(\pi) := \max_{\varphi\in\pi} \mathsf{ldepth}(\varphi)$.
- Quantifier depth: $\mathsf{qdepth}(\pi) := \max_{\varphi\in\pi} \mathsf{qdepth}(\varphi)$.
- Restriction on formulas: *if R is a rule, we can put restrictions on the class of formulas that the rule can be applied to. E.g. we can restrict the* cut formula *in the* cut rule *to belong to bounded-depth formulas.*

In addition, we defined the proof depth *of a proof π, denoted by $\mathsf{depth}(\pi)$, as the depth of π's construction dag. We can extend these complexity measures to proof families in a point-wise manner, e.g. $\mathsf{size}(\{\pi_n\}_n)$ is the function that maps n to $\mathsf{size}(\pi_n)$.*

We will be mainly interested in the relation of these measures to the size of the formulas being proven.

Definition 34 (The proof systems PK, Frege, d-Frege)**.** *Table 3.1 lists the rules for* PK*. We also use* Frege *to refer to* PK*. For $d \in \mathbb{N}$, d-*Frege *is the subsystem of* Frege *obtained by restricting the cut formulas in the cut rule* Cut *to formulas of depth at most d. We call a d-*Frege *proof system a bounded depth proof system and denote them collectively as* bdFrege*.*

Definition 35 (The proof system G)**.** *The proof system* G *is obtained from* Frege *by adding the propositional quantifier introduction rules listed in table 4.1.*

G_i is the subsystem of G *where the cuts are restricted to formulas of quantifier depth i. G_0 is the subsystem of* G *where the cuts are quantifier-free. We call the G_i proof systems G_∞ proof systems.*

*For $d \in \mathbb{N}$, the proof system d-*G *is the subsystem of* G *where the depth of the quantifier-free part of cut formulas is bounded by d. We call these proof systems* bdG *proof systems. For $d, i \in \mathbb{N}$, the proof system d-G_i is the subsystem of G_i where the depth of the quantifier-free part of cut formulas is bounded by d. We call the d-G_i proof systems bdG_i proof systems. A bdG_0 proof system means a d-G_0 proof system for some d. A bdG_∞ proof system means a d-G_i proof system for some d and i.*

Remark 14. *bdG_0, bdG_∞, G_∞ are not proof systems themselves. Viewed as the collection of proof families from their member proof systems they become* proof complexity classes*, see definition 37.*

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. We require the target formula to be $\bot$ or $\top$. Following [CN10], we use the definition of G (and its

$$\frac{\varphi[\vec{q}/\vec{\psi}], \Gamma \Rightarrow \Delta}{\forall \vec{q}\ \varphi, \Gamma \Rightarrow \Delta}\ \forall\mathbf{L} \qquad \frac{\Gamma \Rightarrow \varphi[\vec{q}/\vec{p}], \Delta}{\Gamma \Rightarrow \forall \vec{q}\ \varphi, \Delta}\ \forall\mathbf{R}$$

$$\frac{\varphi[\vec{q}/\vec{p}], \Gamma \Rightarrow \Delta}{\exists \vec{q}\ \varphi, \Gamma \Rightarrow \Delta}\ \exists\mathbf{L} \qquad \frac{\Gamma \Rightarrow \varphi[\vec{q}/\vec{\psi}], \Delta}{\Gamma \Rightarrow \exists \vec{q}\ \varphi, \Delta}\ \exists\mathbf{R}$$

G is obtained by adding the propositional quantifier rules to PK (table 3.1).

Table 4.1: G

$$\frac{\{\varphi_i, \Gamma \Rightarrow \psi_i, \Delta\}_i \qquad \{\psi_i, \Gamma \Rightarrow \varphi_i, \Delta\}_i}{\alpha(\vec{\varphi}), \Gamma \Rightarrow \alpha(\vec{\psi}), \Delta}\ \mathbf{Ext}$$

H is obtained by adding the Ext rule to G (table 4.1).

Table 4.2: H

subsystems) that restricts only the cut formulas. 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.

Remark 15. *It is conservative to extend the class of target formulas to a larger class* Γ *in the presence of cut for formulas in* Γ *and* Ext *rule (table 4.2) if there are function symbols. The proof is similar to [CN10, §VII.3.6, p. 176] and follows from the fact that there are polynomial-size cut-free* Frege *proofs for* $\psi \Rightarrow \varphi[p/\top] \leftrightarrow \varphi[p/\psi]$, $\neg\psi \Rightarrow \varphi[p/\bot] \leftrightarrow \varphi[p/\psi]$. *The only new case that needs to be considered is the function symbol case which follows from the* Ext *rule.*

In our systems, we allow the introduction of a quantifier over multiple propositional variables in a single step. For example, we can derive $\exists \vec{p}\ \varphi$ in a single step from $\varphi[\vec{p}/\vec{\psi}]$. 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 for obtaining a nicer correspondence with first-order proofs.

Definition 36 (The proof system H)**.** *The proof system* H *is the extension of* G *obtained by allowing propositional function symbols and including the extensionality rule* Ext *given in table 4.2 for function symbols.*

Theorem 4. H *is a complete and sound proof system for relativized quantified propositional formulas.*

Proof. The proof is similar to the completeness and soundness proof for QPC(R) in [Coo12].

The soundness of the proof system follows from the soundness of the rules and axioms by induction on the structure of the proof. We only need to check the soundness of Ext. If a formula in Γ is false or a formula in Δ is true we are done. Otherwise, the interpretation of $\vec{\varphi}$ and $\vec{\psi}$ have the same value. Therefore, the interpretation of $\alpha(\vec{\varphi})$ and $\alpha(\vec{\psi})$ have the same value.

Note that by induction on the structure of formulas we can show that $p, \varphi(\top) \Rightarrow \varphi(p)$, $p, \varphi(p) \Rightarrow \varphi(\top)$, $\neg p, \varphi(\bot) \Rightarrow \varphi(p)$, and $\neg p, \varphi(p) \Rightarrow \varphi(\bot)$ have H proofs. If φ is a propositional variable it follows from the axioms and weakening. If φ is an application of a function symbol then it follows from Ext. The induction step is straightforward. As a result both $\forall p \ \varphi(p) \Leftrightarrow \varphi(\top) \wedge \varphi(\bot)$ and $\exists p \ \varphi(p) \Leftrightarrow \varphi(\top) \vee \varphi(\bot)$ have H proofs. Therefore, we can replace quantifiers with conjunctions and disjunctions. If $\vec{\psi}$ is a sequence of n formulas then $\{\tau \leftrightarrow \vec{\psi} \rightarrow \alpha(\tau)\}_{\tau \in \{\top,\bot\}^n} \Leftrightarrow \alpha(\vec{\psi})$ also has an H proof. Therefore, we can restrict over attention to formulas where function symbols are only applies to $\top$ and $\bot$. Note that such a formula with no variables is essentially like a propositional variable.

The completeness follows by induction on the structure of the valid sequents. In the base case there are no logical connectives. If $\mathcal{S}$ is a valid sequent then either its succedent contains $\top$ or its antecedent contains $\bot$ or a formula appears in both of them. In all cases we can derive $\mathcal{S}$ from axioms and weakening rules.

In the induction step there has to be at least a logical connective or a function symbol. If there is a logical connective then we can use the rule for the connective in the reverse direction as the rules for connectives preserve validity in both directions. □

We define the notion of *proof complexity class* in a similar way to nonuniform circuit complexity classes in computational complexity.

Definition 37 (Proof complexity class)**.** *Let $\mathcal{Q}$ be a Frege-style proof system. A $\mathcal{Q}$* proof complexity class *is a set of $\mathcal{Q}$ proof families. Let $\mathcal{F}$ be a $\mathcal{Q}$ proof complexity class. We say a proof family $\{\pi_n\}_n$ is an $\mathcal{F}$-proof for a formula family $\{\varphi_n\}_n$ and write $\{\pi_n\}_n : \mathcal{F} \vdash \{\varphi_n\}_n$ iff for all n, $\pi_n : \mathcal{Q} \vdash \varphi_n$ and $\{\pi_n\}_n \in \mathcal{F}$. We write $\mathcal{F} \vdash \{\varphi_n\}_n$ when $\{\varphi_n\}_n$ has an $\mathcal{F}$-proof.*

Size, depth, and quantifier depth are the main measures we use to define proof complexity classes.

Definition 38 ($\mathcal{Q}$-Size(s), $\mathcal{Q}$-DepthSize(d, s))**.** *Let $\mathcal{Q}$ be a subsystem of* H *and s and d be two functions over natural numbers.*

- *The proof complexity class $\mathcal{Q}$-*Size(s) *is defined as the set of $\mathcal{Q}$-proof families $\{\pi_n\}_n : \mathcal{Q} \vdash \{\varphi_n\}_n$ where* $\mathsf{size}(\pi_n) \leq s(\mathsf{size}(\varphi_n))$.

- *The proof complexity class* $\mathcal{Q}$-$\mathsf{DepthSize}(d,s)$ *is defined as the set of* $\mathcal{Q}$*-proof families* $\{\pi_n\}_n : \mathcal{Q} \vdash \{\varphi_n\}_n$ *where* $\mathsf{size}(\pi_n) \leq s(\mathsf{size}(\varphi_n))$ *and the depth of the cut formulas in* π_n *is at most* $d(\mathsf{size}(\varphi_n))$.

Remark 16. *Note that the classes defined above are also proof systems.*

Convention 4. *We write* $\mathsf{Frege}(d,s)$ *for* $\mathsf{Frege}\text{-}\mathsf{DepthSize}(d,s)$.

Example 8. *The set of* Frege *proof families is a proof complexity class.*

Example 9. *The set of* bdFrege *proof families is a proof complexity class. Similarly, the set of* bdG_0, bdG_∞, *and* G_∞ *proof families are proof complexity classes.*

Example 10. *The set of polynomial-size* Frege *proof families is a proof complexity class.*

Example 11. *The set of polynomial-size bounded-depth* Frege *proof families is a proof complexity class.*

Example 12. *The set of subexponential-size bounded-depth* Frege *proof families is a proof complexity class.*

The notion of an arbitrary proof class is very general and hard to work with as is the notion of an arbitrary complexity class. Similar to computational complexity theory, we focus on complexity classes which are obtained by restricting nice complexity measures like those mentioned above.

Definition 39 (Nice proof complexity class)**.** *We say a proof class is* nice *if it is a union of* $\mathcal{Q}$-DepthSize *classes of some Frege-style proof system* $\mathcal{Q}$.

We define the following nice proof classes:

Definition 40 ($\mathsf{Frege}(O(1), n^{O(1)})$)**.** *The class of polynomial-size bounded-depth* Frege *proofs is defined as*

$$\mathsf{Frege}(O(1), n^{O(1)}) = \bigcup_{s \in \mathsf{poly}, d \in \mathbb{N}} \mathsf{Frege}(d,s).$$

Remark 17. *The proof complexity class polynomial-size* Frege *is also referred to as* NC^1-Frege *in the literature. We will reserve* NC^1-Frege *to refer to the subclass of* CFrege *[Jeř05] where lines in the proof are* NC^1 *circuits. An important subclass of polynomial-size* Frege *is polynomial-size bounded-depth* Frege, *which is obtained by restricting the depth of cut formulas in the proof to be* $O(1)$. *It is also referred to as* AC^0-Frege *in the literature, however, we will reserve* AC^0-Frege *to refer to the subclass of* CFrege *where lines in the proof are* AC^0 *circuits.*

Remark 18. *Note that nice proof complexity classes are closed under a concrete substitution, i.e., if we replace all occurrences of a free variable in a proof family that belongs to a nice proof complexity class with* $\bot$ *(or* $\top$*), then the resulting family also belongs to that proof class.*

4.3 Proof Reductions and Universal Tautologies

Definition 41 (Proof reduction)**.** *Let $\mathcal{R}$ be a Frege-style proof complexity class, Γ a set of formula families, and $\{\varphi_n\}_n$ a formula family. We say $\{\varphi_n\}_n$ is* $\mathcal{R}$-reducible *to Γ and write $\{\varphi_n\}_n \leq^{\mathcal{R}} \Gamma$ when $\Gamma, \mathcal{R} \vdash \{\varphi_n\}_n$.*

In this way, we can associate with every proof class a *proof reduction* class.

Remark 19. *The reductions defined above are similar to the oracle Turing reductions in computational complexity which are defined using a reduction function. As in complexity theory, it is possible to define more general notions of reducibility that work for more general proof systems. For example, we define many-one reductions in computational complexity theory. Similarly, we can define a weaker more general form of reduction for arbitrary proof systems where an $\mathcal{R}$-reduction from $\{\varphi_n\}_n$ to $\{\psi_n\}_n$ is a proof $\mathcal{R} \vdash \{\varphi_{f(n)} \to \psi_n\}_n$ for some polynomially-bounded function f.*

Definition 42 (Simulation)**.** *We say that a proof system $\mathcal{Q}'$ f-simulates another proof system $\mathcal{Q}$ iff for every proof in $\mathcal{Q}$ there is proof in $\mathcal{Q}'$ for the same tautology and the size of the $\mathcal{Q}'$ proof is at most f of the size of the $\mathcal{Q}$ proof. We say that a proof system $\mathcal{Q}'$ (effectively)* p-simulates *another proof system $\mathcal{Q}$ iff there is a polynomial-time computable function that maps the proofs in $\mathcal{Q}$ to proofs of the same tautology in $\mathcal{Q}'$.*

Remark 20. *Effective proof reductions are similar to Levin's reductions [Lev73] in computational complexity theory: a Levin reduction from an* NP *problem with verifier $\mathcal{Q}$ to an* NP *problem with verifier $\mathcal{Q}'$ is a polynomial-time function R_i such that for all φ and π, $\mathcal{Q}$ accepts φ and π iff $\mathcal{Q}'$ accepts φ and $R_i(\pi)$.*

We also have the many-one reductions between search problems. Let $\mathcal{Q}$ and $\mathcal{Q}'$ be two NP *search problems. We say that $\mathcal{Q}$ is many-one reducible to $\mathcal{Q}'$ if there are functions R_i and R_o such that for all $\varphi \in \mathsf{dom}(\mathcal{Q})$, $R_i(\varphi) \in \mathsf{dom}(\mathcal{Q}')$ and for all $\pi \in \mathcal{Q}'(R_i(\varphi))$, $R_o(\varphi, \pi) \in \mathcal{Q}(\varphi)$. If we add the condition that R_i and R_o are polynomially bounded, we essentially obtain the general definition of p-simulation between proof systems from [Rec76, pp. 113–114]. Reckhow considers additional conditions on the query $R_i(\varphi)$ to $\mathcal{Q}'$: it needs to be logically equivalent to φ and preserve the logical structure of the formula by commuting with substitution [Rec76, pp. 124–125, 138–139].*

These reductions are still many-one. We can define reductions between search problems even more generally. See e.g. [BCE+95] which considers more general oracle reductions using type-2 complexity theory [KC96]. All of these reductions can be used to compare proof systems since a proof system can be viewed as a proof search problem.

We will define universality as an analog of completeness in computational complexity theory:

Definition 43 (Universal formula family)**.** *Let* Γ *be a set of formula families,* $\mathcal{F}$ *a proof complexity class, and* $\mathcal{R}$ *a class of proof reductions. We say* Γ *is* universal in $\mathcal{F}$ *w.r.t.* $\mathcal{R}$ *iff*

- Γ *belongs to* $\mathcal{F}$*: All formula families in* Γ *are provable in* $\mathcal{F}$*.*
- Γ *is* universal for $\mathcal{F}$*: Any formula family in* $\mathcal{F}$ *is* $\mathcal{R}$*-reducible to concrete instances (definition 28) of* Γ*.*

Remark 21. *In general, we can get an (effective) p-simulation of a system* $\mathcal{P}$ *by a sufficiently strong proof system* $\mathcal{Q}$ *when* $\mathcal{Q}$ *has polynomial-size proofs of a universal class of formula families for* $\mathcal{P}$*.*

4.4 Definability of Truth in Proof Classes

Let $\mathcal{C}$ be a formula complexity class. Let $\ulcorner . \urcorner : \mathcal{C} \to \{0,1\}^*$ be an encoding of formulas in $\mathcal{C}$ similar to [CN10]. We use "$\tau \vDash \varphi$" to denote a polynomial-size formula expressing that the truth assignment encoded by τ satisfies the formula encoded by φ, possibly using helper variables to encode the computation of a suitable evaluation algorithm [Kra12]. See chapter 6 for a demonstration in the uniform setting. A similar construction works in the nonuniform setting.

Definition 44. *We say that the truth for formula complexity class* $\mathcal{C}$ *is definable in a proof complexity class* $\mathcal{F}$ *iff there is a formula family* $\{\text{“}\tau \vDash_{n,m} \varphi\text{”}\}_{n,m}$ *such that* $\mathcal{F}$ *proves the inductive Tarski definition of truth for encoded formulas of size* n *in* $\mathcal{C}$*. In other words,* $\mathcal{F}$ *proves the conjunction of:*

$$\text{“}\varphi = \top\text{”} \to (\text{“}\vDash \varphi\text{”})$$

$$\text{“}\varphi = \bot\text{”} \to (\text{“}\nvDash \varphi\text{”})$$

$$\text{“}\varphi = p\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \text{“}\tau(p) = \top\text{”})$$

$$\text{“}\varphi = \alpha(\vec{p})\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \text{“}\alpha(\tau(\vec{p})) = \top\text{”})$$

$$\text{“}\varphi = \neg\varphi_0\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \text{“}\tau \nvDash \varphi_0\text{”})$$

$$\text{“}\varphi = \varphi_0 \wedge \varphi_1\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \text{“}\tau \vDash \varphi_0\text{”} \wedge \text{“}\tau \vDash \varphi_1\text{”})$$

$$\text{“}\varphi = \varphi_0 \vee \varphi_1\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \text{“}\tau \vDash \varphi_0\text{”} \vee \text{“}\tau \vDash \varphi_1\text{”})$$

$$\text{“}\varphi = \exists p\ \varphi_0\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \exists p\ \text{“}\tau[p \mapsto p] \vDash \varphi_0\text{”})$$

$$\text{“}\varphi = \forall p\ \varphi_0\text{”} \to (\text{“}\tau \vDash \varphi\text{”} \leftrightarrow \forall p\ \text{“}\tau[p \mapsto p] \vDash \varphi_0\text{”})$$

where φ stands for a member of $\mathcal{C}$, α stands for a member of $\mathcal{L}$, and p for a member of $\mathcal{V}$.

Remark 22. *Some of the cases above can be unnecessary if $\mathcal{C}$ does not need them.*

4.5 Soundness Tautologies

In this section we prove that the soundness statements for nice proof complexity classes are universal for them. This is similar to the completeness of the circuit evaluation problems for nice circuit complexity classes. We first show that it is possible to derive the truth of a formula in a proof class from the soundness statement for the proof class. In the second step, we derive a formula from its truth. The arguments in this and following sections are similar to the classical arguments about the relation of proof systems extending EFrege and their soundness [Kra95, §14.1].

Definition 45 (Soundness of a nice proof class)**.** *Let the formula $\mathsf{Snd}(\mathcal{Q}\text{-}\mathsf{DepthSize}(d,s))_n$ denote the soundness of $\mathcal{Q}$-$\mathsf{DepthSize}(d,s)$ proofs of size n:*

$$\text{“}\pi : \mathcal{Q}\text{-}\mathsf{DepthSize}(d,s) \vdash \varphi\text{”} \Rightarrow \text{“}\tau \vDash \varphi\text{”}$$

where π, φ, and τ each denotes n free propositional variables. Let $\mathsf{Snd}(\mathcal{Q}\text{-}\mathsf{DepthSize}(d,s))$ stand for the family $\{\mathsf{Snd}(\mathcal{Q}\text{-}\mathsf{DepthSize}(d,s))_n\}_n$, the soundness formula family for proof families in $\mathcal{Q}$-$\mathsf{DepthSize}(d,s)$.

Let $\mathcal{F}$ be a nice proof class obtained from taking the union of some $\mathcal{Q}$-$\mathsf{DepthSize}(d,s)$ proof classes. The soundness of $\mathcal{F}$, denoted by $\mathsf{Snd}(\mathcal{F})$ is defined as

$$\{\mathsf{Snd}(\mathcal{Q}\text{-}\mathsf{DepthSize}(d,s)) \mid \mathcal{Q}\text{-}\mathsf{DepthSize}(d,s) \in \mathcal{F}\}.$$

Example 13. *Consider $\mathsf{Frege}(O(1), n^{O(1)})$. $\mathsf{Snd}(\mathsf{Frege}(O(1), n^{O(1)}))$ stands for the set of soundness formula families of $\mathsf{Snd}(\mathsf{Frege}(d,s))$ where $d \in \mathbb{N}, s \in \mathsf{poly}(n)$:*

$$\mathsf{Snd}(\mathsf{Frege}(O(1), n^{O(1)})) = \{\mathsf{Snd}(\mathsf{Frege}(d,s)) \mid d \in \mathbb{N}, s \in \mathsf{poly}(n)\}.$$

4.6 Universality of Soundness

Definition 46 (Closure under cuts)**.** *We say a proof class $\mathcal{R}$ is* closed under Φ-cuts *iff whenever there are $\mathcal{R}$-proofs for $\Gamma \Rightarrow \varphi, \Delta$ and $\Sigma, \varphi \Rightarrow \Pi$ and $\varphi \in \Phi$, there is an $\mathcal{R}$ proof for $\Gamma, \Sigma \Rightarrow \Delta, \Pi$.*

Definition 47. *Let $\mathcal{R}$ denote a class of proof reductions. We say $\mathcal{R}$ is* reasonable *iff*

- *Every closed Σ_0^q formula family has a polynomial-size proof family in $\mathcal{R}$.*
- *$\mathcal{R}$ is closed under composition, i.e. $\mathcal{T}, \mathcal{R} \vdash \mathcal{S}$ whenever $\mathcal{T}, \mathcal{R} \vdash \mathcal{T}'$ and $\mathcal{T}', \mathcal{R} \vdash \mathcal{S}$.*

If in addition $\mathcal{R}$ is closed under Φ-cuts, we say that $\mathcal{R}$ is Φ-reasonable.

Note that the proof class polynomial-size cut-free PK is reasonable. We will generally be interested in classes containing polynomial-size cut-free PK so we only need to check if they are closed under composition.

Definition 48 (Weak universality)**.** *A class of formula families Γ is* weakly universal *for a proof class $\mathcal{F}$ w.r.t. a class of proof reductions $\mathcal{R}$ iff the* truth *of every formula provable in $\mathcal{F}$ has an $\mathcal{R}$ proof from concrete instances of formulas in Γ. In other words, for every $\{\varphi_n\}_n$, if $\mathcal{F} \vdash \{\varphi_n\}_n$ then $\overline{\Gamma} + \mathcal{R} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$.*

Definition 49 (Strong universality)**.** *A class of formula families Γ is* (strongly) universal *for a proof class $\mathcal{F}$ w.r.t. a class of proof reductions $\mathcal{R}$ iff the every formula provable in $\mathcal{F}$ has an $\mathcal{R}$ proof from concrete instances of formulas in Γ. In other words, for every $\{\varphi_n\}_n$, if $\mathcal{F} \vdash \{\varphi_n\}_n$ then $\overline{\Gamma} + \mathcal{R} \vdash \varphi$.*

Remark 23. *Weak universality is an important notion and has the essence of universality. It is sometimes required to prove an equivalent formula in place of the original formula because it is not possible to express the formula directly in the language of the stronger system or it is not possible to prove the equivalence of the formula and its truth. The simulation of a Frege system using $\{\leftrightarrow, \bot\}$ by one using $\{\neg, \wedge\}$. The method of indirect translations in [Rec76] essentially evaluates formulas in the second language using Spira's method [Spi71] to express them in the first language. The argument is simpler if we first define the truth of formulas from the second language in the first language.*

Remark 24. *We can use a notion of reasonable reductions which is weaker than polynomial-size bounded-depth* PK*. However, since the weakest class we consider here is polynomial-size bounded-depth* PK *corresponding to* AC^0 *we use this simpler definition.*

Recall that a closed formula is a formula with no free variables.

Lemma 1. *Every closed quantifier-free propositional tautology has a polynomial-size cut-free* Frege *proof.*

Proof. By induction on the structure of the formula. □

Theorem 5 (Soundness is weakly universal)**.** *Let $\mathcal{F}$ be a nice proof class. Then* $\mathsf{Snd}(\mathcal{F})$ *is weakly universal for $\mathcal{F}$ with respect to polynomial-size bounded-depth* Frege *reductions.*

Proof. Let $\mathcal{F}$ be a nice proof class and $\mathcal{R}$ be poly-size bounded-depth Frege. Assume that $\mathcal{F} \vdash \{\varphi_n\}_n$. We want to show that $\overline{\mathsf{Snd}(\mathcal{F})}, \mathcal{R} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$.

Since $\mathcal{F} \vdash \{\varphi_n\}_n$ there is $\mathcal{Q} \in \mathcal{F}$ such that $\mathcal{Q} \vdash \{\varphi_n\}_n$. Therefore, there is a $\{\pi_n\}_n$ such that $\{\pi_n\}_n : \mathcal{Q} \vdash \{\varphi_n\}_n$. Now by lemma 1 $\mathcal{R} \vdash \{\text{“}\ulcorner \pi_n \urcorner \vdash \ulcorner \varphi_n \urcorner\text{”}\}_n$ as it is a closed formula. Now since $\mathcal{R}$ is closed under bounded-depth cuts and $\{\text{“}\ulcorner \pi_n \urcorner \vdash \ulcorner \varphi_n \urcorner\text{”}\}_n$ is a bounded-depth formula we can cut it with the soundness for π_n and φ_n: $\mathsf{Snd}(Q)[\pi, \varphi/\ulcorner \pi_n \urcorner, \ulcorner \varphi_n \urcorner]$ which is in $\overline{\mathsf{Snd}(\mathcal{F})}$. Therefore, we obtain a proof of $\overline{\mathsf{Snd}(\mathcal{F})}, \mathcal{R} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$. □

Strong universality is obtained if we can prove that the truth of formulas implies them.

Definition 50 (Semantic reflection property)**.** *We say a proof class $\mathcal{R}$ has the* semantic reflection property *for Φ-formulas when there is an $\mathcal{R}$-reduction from “$\vec{r} \vDash \ulcorner \varphi \urcorner$” to $\varphi[/\vec{r}]$ for every $\varphi \in \Phi$.*

Corollary 1. *Let $\mathcal{R}$ be a reasonable class of proof reductions and Γ be weakly universal for $\mathcal{F}$. If $\mathcal{R}$ has the semantic reflection property, then Γ is strongly universal for $\mathcal{F}$ w.r.t. $\mathcal{R}$.*

Remark 25. *If a proof class can prove Tarski's inductive definition for a truth formula, we can use structural induction on the structure of the formulas to prove the semantic reflection property of the truth formula.*

Next, we turn our attention to the question of provability of the soundness formulas.

Definition 51 (Soundness of a rule)**.** *Let R be a rule of a proof system with k top sequents. The soundness of R states that if π is an instance of R and the top sequents in R are true under a truth assignment τ, then the bottom sequent in R is true under τ. We say $\mathcal{F}$ proves the soundness of R when there is an $\mathcal{F}$ proof for “$\tau \vDash bot(\pi)$” from “$\pi \in R$” and “$\tau \vDash top(R)$”.*

Lemma 2. *Polynomial-size bounded-depth* PK *proves the soundness of the rules of cut-free* PK.

Proof. The proof is essentially a case analysis. □

Lemma 3. *Polynomial-size bounded-depth* PK *proves the soundness of the depth d cut rule.*

Proof. The proof again is by case analysis based on the truth of falsity of the cut formula. □

Theorem 6. *Polynomial-size bounded-depth* Frege *proves its own soundness.*

Proof. The truth of depth d formulas can be expressed using bounded-depth formulas. The construction of the proof proceeds by induction over the structure of the proof using the provability of the soundness of the rules. □

For other proof classes we only need to prove the soundness of the appended rules. This will be usually a cut rule. However, the real key to proving the soundness of a proof is the ability to express the truth of the formulas in the proof under a given truth assignment for the parameters of the proof. If we can express the truth of the lines in the proofs, we can carry out induction on the structure of the proof using them. Limiting the cut rule restricts the class of formulas that can appear in the proof since every formula in a proof is a subformula of either a cut formula or the end-sequent.

Definition 52. *We say that proof complexity class $\mathcal{F}$* weakly contains *proof complexity class $\mathcal{F}'$ and write $\mathcal{F}' \leq \mathcal{F}$ when for every $\{\varphi_n\}_n$, $\mathcal{F}' \vdash \{\varphi_n\}_n$ implies $\mathcal{F} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$.*

Theorem 7. *Let $\mathcal{F}$ be a proof complexity class containing $\mathcal{R}$. Let LOF$'$ be a nice proof complexity class. If $\mathcal{F}$ proves $\overline{\mathsf{Snd}(\mathcal{F}')}$, then $\mathcal{F}' \leq \mathcal{F}$.*

Proof. Assume that $\mathcal{F}' \vdash \{\varphi_n\}_n$. By theorem 5, $\mathsf{Snd}(\mathcal{F}')$ is weakly universal for $\mathcal{F}'$ with respect to $\mathcal{R}$. Therefore, $\overline{\mathsf{Snd}(\mathcal{F}')}, \mathcal{R} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$. Since $\mathcal{F} \vdash \overline{\mathsf{Snd}(\mathcal{F}')}$ and $\mathcal{F} \vdash \mathcal{R}$ we have $\mathcal{F} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$. □

Chapter 5

From Uniform to Nonuniform: Propositional Translation

In this chapter we provide *propositional translations* from proofs in uniform io-typed theories (chapter 3) to propositional proof families in nonuniform H-proof complexity classes (chapter 4). Propositional translations were introduced in [Coo75]. We extend the propositional translation of two-sorted theories from [CN10] to our io-typed theories. The concept of propositional translations is analogous to the Cook-Levin theorem [Coo71] in computational complexity theory which connects the uniform computation model of Turing machines to the nonuniform computation model of circuits.

A propositional translation consists of translating formulas and proofs of formulas. We begin by defining a propositional translation from first-order formulas to relativized quantified propositional formulas in section 5.1. In section 5.2 we define the propositional translation of proofs. Our focus will be on proofs in ioV^0. In section 5.3 we discuss the translation of theories built upon ioV^0. The essential idea of our translation is that to translate a theory built on top of ioV^0 we can first Skolemize the theory and then translate proofs to proof families in H plus axioms. Afterward, we can provide explicit witnessing formulas for the function symbols of the axioms and explicit proof families for axioms with function symbols replaced with explicit witnesses. The first part is general and independent of particular theories being translated.

5.1 Translating Formulas

We first define the propositional translation for (io-type two-sorted) first-order terms. A first-order term will be translated into a family of sequences of propositional formulas.

The family is indexed by the size of the free variables in the term. The number of formulas in each sequence is determined by the size of the term. Each formula in the sequence corresponds to one bit of the term. A first-order bounded formula will be translated to a family of relativized quantified propositional formulas.

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 first-order function symbol. The number of propositional function symbols is the length of the first-order function symbol.

Definition 53. *If $\mathcal{X}$ is a sequence, $(\mathcal{X})_i$ denotes the ith item in the sequence $\mathcal{X}$. If $\vec{p} = (p_0, \ldots, p_{k-1})$ is a sequence of propositional variables, then $(\vec{p})_i$ is p_i for $i < k$. We consider $(\vec{p})_i$ to be $\bot$ when $i \geq k$.*

Definition 54 (Translation context)**.** *Let $\mathcal{V}$ be the set of variables. A* translation context *is a function $\sigma : \mathcal{V} \to \mathbb{N}$ that determines the size of variables by assigning a natural number to each variable.*

Remark 26. *For a number variable, the translation context determines the value of the variable.*

Definition 55 (Function modification)**.** *If σ is a translation context, then $\sigma[\vec{x} \mapsto \vec{n}]$ denotes the function obtained from σ by mapping $\vec{x}$ to $\vec{n}$:*

$$\sigma[\vec{x} \mapsto \vec{n}](y) = \begin{cases} n_i & y = x_i \\ \sigma(y) & o.w. \end{cases}$$

Notation 10. *A propositional translation is a function that maps first-order terms and formulas with a translation context σ to (a sequence of) propositional formulas. Let $[\![\cdot]\!]$ be a translation function and σ a translation context. The translation of a first-order term t under $[\![\cdot]\!]$ and σ denoted by $[\![t]\!]_\sigma$ is a sequence of propositional formulas where each formula in the sequence corresponds to one bit of t. The translation of a first-order formula φ under $[\![\cdot]\!]$ and σ is a propositional formula denoted by $[\![\varphi]\!]_\sigma$.*

Convention 5. *We may simply write $[\![t]\!]_{\vec{n}}$ 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 are being mapped to $\vec{n}$.*

The translation context determines the size of the variables. We need to extend the translation context to all terms in the language. We will use σ to determine the size of

$$
\begin{aligned}
\sigma(0) &:= 0 \\
\sigma(1) &:= 1 \\
\sigma(|T|) &:= \sigma(T) \\
\sigma(t+s) &:= \sigma(t) + \sigma(s) \\
\sigma(t{\cdot}s) &:= \sigma(t){\cdot}\sigma(s) \\
\sigma(\mathsf{pd}(t)) &:= \mathsf{pd}(\sigma(t)) \\
\sigma(f(\vec{t},\vec{T})) &:= |f|(\sigma(\vec{t}),\sigma(\vec{T})) \\
\sigma(F(\vec{t},\vec{T})) &:= |F|(\sigma(\vec{t}),\sigma(\vec{T}))
\end{aligned}
$$

Table 5.1: Extended Translation Context

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 function symbols in our first-order languages have explicit sizes in terms of the size of their inputs[1]. The size of a k-ary function symbol is a function from $\mathbb{N}^k$ to $\mathbb{N}$. We denote the size of a function symbol F by $|F|$. We also extend the translation context over sequences in a point-wise manner, i.e. $\sigma(t_k,\ldots,t_0) := \sigma(t_k),\ldots,\sigma(t_0)$.

Definition 56 (Extended translation context)**.** *The extended translation context of σ is given in table 5.1. For a k-ary term t, $\sigma(t)$ is a function from $\mathbb{N}^k$ to $\mathbb{N}$.*

Recall that each function symbol in our language has a size bound and the size of function symbol F is denoted by $|F|$. In our theories the size of function symbols are polynomially bounded.

Lemma 4. *Assume that all function symbols of the language have polynomial size. For any term t, $\sigma(t)$ is bounded by a polynomial in σ of variables in t.*

Proof. By induction over the structure of the terms. □

Terms are translated recursively: a number term t is translated to a sequence of size $\sigma(t)+1$, and a string term T is translated to a sequence of size $\sigma(T)$. A unary number n is represented by $\top\bot^n$, i.e. a sequence of size $n+1$ where only the nth bit is $\top$. We index bits from the right, e.g. the rightmost bit has index 0. We extend the translation to sequences in a point-wise manner, i.e. $[\![t_k,\ldots,t_0]\!]_\sigma := [\![t_k]\!]_\sigma,\ldots,[\![t_0]\!]_\sigma$.

We can use any reasonable AC^0 formula for the translation of the functions of $\mathcal{L}_2$. The main requirement is that the translated axioms of io2Basic must have simple propositional proofs.

[1] See definition 10 and the discussion following it.

$$[\![0]\!]_\sigma := (\top) \qquad$$

$$[\![1]\!]_\sigma := (\top, \bot) \qquad$$

$$[\![x]\!]_\sigma := (\top, \overbrace{\bot, \ldots, \bot}^{\sigma(x) \text{ times}}) \qquad ([\![x]\!]_\sigma)_k := \begin{cases} \top & k = \sigma(x) \\ \bot & o.w. \end{cases}$$

$$[\![|T|]\!]_\sigma := (\top, \overbrace{\bot, \ldots, \bot}^{\sigma(T) \text{ times}}) \qquad ([\![|T|]\!]_\sigma)_k := \begin{cases} \top & k = \sigma(T) \\ \bot & o.w. \end{cases}$$

$$[\![X]\!]_\sigma := (p^X_{\sigma(X)-1}, \ldots, p^X_0) \qquad ([\![X]\!]_\sigma)_k := p^X_k$$

$$[\![s+t]\!]_\sigma := (o_{\sigma(s+t)}, \ldots, o_0) \quad \text{where } o_k \text{ is} \quad ([\![s+t]\!]_\sigma)_k := \bigvee_{\substack{i \le \sigma(s) \\ j \le \sigma(t) \\ i+j=k}} ([\![s]\!]_\sigma)_i \wedge ([\![t]\!]_\sigma)_j$$

$$[\![s \cdot t]\!]_\sigma := (o_{\sigma(s \cdot t)}, \ldots, o_0) \quad \text{where } o_k \text{ is} \quad ([\![s \cdot t]\!]_\sigma)_k := \bigvee_{\substack{i \le \sigma(s) \\ j \le \sigma(t) \\ i \cdot j=k}} ([\![s]\!]_\sigma)_i \wedge ([\![t]\!]_\sigma)_j$$

$$[\![\mathsf{pd}(t)]\!]_\sigma := (o_{\sigma(\mathsf{pd}(t))}, \ldots, o_0) \quad \text{where } o_k \text{ is} \quad ([\![\mathsf{pd}(t)]\!]_\sigma)_k := \begin{cases} (\bigvee_{i=0,1} [\![t]\!]_\sigma)_i & k = 0 \\ ([\![t]\!]_\sigma)_{k+1} & o.w. \end{cases}$$

$$[\![f(\vec{t}, \vec{T})]\!]_\sigma := (\alpha^{f,\sigma,\sigma(f(\vec{t},\vec{T}))}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma), \ldots, \alpha^{f,\sigma,0}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma)) \qquad ([\![f(\vec{t}, \vec{T})]\!]_\sigma)_k := \alpha^{f,\sigma,k}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma)$$

$$[\![F(\vec{t}, \vec{T})]\!]_\sigma := (\alpha^{F,\sigma,\sigma(F(\vec{t},\vec{T}))-1}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma), \ldots, \alpha^{F,\sigma,0}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma)) \qquad ([\![F(\vec{t}, \vec{T})]\!]_\sigma)_k := \alpha^{F,\sigma,k}([\![\vec{t}]\!]_\sigma, [\![\vec{T}]\!]_\sigma)$$

Table 5.2: Propositional Translation of Terms

Definition 57 (Translation of terms)**.** *The translation of terms under σ is given in table 5.2.*

Remark 27. *Note that our translation is slightly more complicated than the translation in [CN10] where the value of number terms is determined by σ. This is not true in general. E.g. consider a number-valued function of strings like* $\mathsf{msb}(X)$ the most significant bit. *The value of* $\mathsf{msb}(X)$ *depends not only on σ but also on the value of X's bits.*

Example 14. *Consider the term $x + y$ with the translation context $\sigma = [x \mapsto 3][y \mapsto 5]$.*

- $[\![x]\!]_\sigma = (\top, \bot, \bot, \bot)$.
- $[\![y]\!]_\sigma = (\top, \bot, \bot, \bot, \bot, \bot)$.
- $([\![x+y]\!]_\sigma)_2 = \bigvee_{\substack{i\leq 3, j\leq 5 \\ i+j=2}} ([\![x]\!]_\sigma)_i \wedge ([\![y]\!]_\sigma)_j = (\bot \wedge \bot) \vee (\bot \wedge \bot) \vee (\bot \wedge \bot) \vee (\bot \wedge \bot)$.
- $([\![x+y]\!]_\sigma)_3 = \bigvee_{\substack{i\leq 3, j\leq 5 \\ i+j=3}} ([\![x]\!]_\sigma)_i \wedge ([\![y]\!]_\sigma)_j = (\bot \wedge \bot) \vee (\bot \wedge \bot) \vee (\bot \wedge \bot) \vee (\top \wedge \bot)$.
- $([\![x+y]\!]_\sigma)_8 = \bigvee_{\substack{i\leq 3, j\leq 5 \\ i+j=8}} ([\![x]\!]_\sigma)_i \wedge ([\![y]\!]_\sigma)_j = (\top \wedge \top)$.

Lemma 5 (Size and depth of translated terms)**.** *For any term t,* $\mathsf{size}([\![t]\!]_\sigma)$ *is bounded by a polynomial in σ. In addition, the translation of terms have a constant depth, i.e.* $\mathsf{ldepth}([\![t]\!]_\sigma) = O(1)$.

Proof. By induction over the structure of the terms. □

Remark 28. *Let F be a string-valued function defined by a monotone non-decreasing term t for its size and formula φ for its bit-graph (section 3.2):*

- $|F(\vec{x}, \vec{X})| := t(\vec{x}, |\vec{X}|)$,
- $y \in F(\vec{x}, \vec{X}) := \varphi(\vec{x}, \vec{X}, y)$.

We can translate F directly without using new propositional function symbols:

- $\sigma(F(\vec{t}, \vec{T})) := t(\sigma(\vec{t}), \sigma(\vec{T}))$,
- $([\![F(\vec{t}, \vec{T})]\!]_\sigma)_k := [\![\varphi(\vec{t}, \vec{T}, y)]\!]_{\sigma[y\mapsto k]}$.

Example 15. *If we include the substring function $T[s,t]$ in the language defined in definition 13, we can translate it using:*

- $\sigma(T[s,t]) := \sigma(t)$,

- $(\llbracket T[s,t] \rrbracket_\sigma)_k := \llbracket x < t \wedge s + x \in T \rrbracket_{\sigma[x \mapsto k]}$.

Remark 29. *Let f is a number-valued function defined by a monotone non-decreasing term t and a formula φ:*

- $f(\vec{x}) \leq t(\vec{x})$,
- $f(\vec{x}) = y := \varphi(\vec{x}, y)$.

We can translate f directly without using new propositional function symbols:

- $\sigma(f(\vec{t})) := t(\sigma(\vec{t}))$,
- $(\llbracket f(\vec{t}) \rrbracket_\sigma)_k := \llbracket \varphi(\vec{t}, y) \rrbracket_{\sigma[y \mapsto k]}$.

The translation works because in our treatment of unary numbers the size of a unary number by itself does not determine its value. In the propositional setting the size of the output is determined by the size of the input. If the size of a unary number determined its value, the output of a number-valued function could depend only on the size of its inputs. However, in our treatment, a unary number of size $n+1$ can correspond to any natural number from 0 to n. Therefore, we can include in our language number function like msb *(the most-significant bit) whose value is not determined by the size of its inputs.*

Example 16. *If we include the most significant bit function* msb *in the language, we can translate it using:*

- $\sigma(\mathsf{msb}(T)) := \sigma(|T|)$,
- $(\llbracket \mathsf{msb}(T) \rrbracket_\sigma)_k := \llbracket y \in T \wedge \forall x < |T| \ \ (x \in T \rightarrow x \leq y) \rrbracket_{\sigma[y \mapsto k]}$.

Example 17. *If we include the fractional power function $x^{\frac{1}{d}}$ in the language defined in definition 14, we can translate it using:*

- $\sigma(t^{\frac{1}{d}}) := \sigma(t)$,
- $(\llbracket t^{\frac{1}{d}} \rrbracket_\sigma)_k := \llbracket y^d \leq t \wedge t < (y+1)^d \rrbracket_{\sigma[y \mapsto k]}$.

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 connectives 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. 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 will be translated as such.

$$
\begin{aligned}
[\![s = t]\!]_\sigma &:= \bigvee_{i \le \min(\sigma(s),\sigma(t))} ([\![s]\!]_\sigma)_i \wedge ([\![t]\!]_\sigma)_i \\
[\![s \le t]\!]_\sigma &:= \bigvee_{\substack{i \le \sigma(s) \\ i \le j \le \sigma(t)}} ([\![s]\!]_\sigma)_i \wedge ([\![t]\!]_\sigma)_j \\
[\![t \in T]\!]_\sigma &:= \bigvee_{i \le \min(\sigma(T),\sigma(t))} ([\![T]\!]_\sigma)_i \wedge ([\![t]\!]_\sigma)_i \\
[\![\bot]\!]_\sigma &:= \bot \\
[\![\top]\!]_\sigma &:= \top \\
[\![\neg\varphi]\!]_\sigma &:= \neg[\![\varphi]\!]_\sigma \\
[\![\psi \wedge \varphi]\!]_\sigma &:= [\![\psi]\!]_\sigma \wedge [\![\varphi]\!]_\sigma \\
[\![\psi \vee \varphi]\!]_\sigma &:= [\![\psi]\!]_\sigma \vee [\![\varphi]\!]_\sigma \\
[\![\exists x \le t \ \ \varphi]\!]_\sigma &:= \bigvee_{i \le \sigma(t)} [\![x \le t \wedge \varphi]\!]_{\sigma[x \mapsto i]} \\
[\![\forall x \le t \ \ \varphi]\!]_\sigma &:= \bigwedge_{i \le \sigma(t)} [\![x \le t \to \varphi]\!]_{\sigma[x \mapsto i]} \\
[\![\exists X = t \ \ \varphi]\!]_\sigma &:= \exists [\![X]\!]_\tau \ \ [\![X = t \wedge \varphi]\!]_\tau \\
[\![\forall X = t \ \ \varphi]\!]_\sigma &:= \forall [\![X]\!]_\tau \ \ [\![X = t \to \varphi]\!]_\tau
\end{aligned}
$$

In the translation of string quantifiers $\tau = \sigma[X \mapsto \sigma(t)]$.

Table 5.3: Propositional Translation of Formulas

Definition 58 (Translation of formulas)**.** *The translation of bounded formulas under σ is given in table 5.3.*

Lemma 6. *For any formula φ, $\mathsf{size}([\![\varphi]\!]_\sigma)$ is bounded by a polynomial in σ. In addition, the translation of bounded formulas have a constant depth, i.e. $\mathsf{ldepth}([\![\varphi]\!]_\sigma) = O(1)$ and $\mathsf{qdepth}([\![\varphi]\!]_\sigma) = O(1)$. Furthermore, the number of quantified propositional variables is equal to the sum of σ of the bounding terms for the bounded string quantifiers.*

Proof. By induction over the structure of the formulas. □

Remark 30. *Later we will look at the relation between the size of the proofs and the size of the formulas. We would consider the size of a translated formula φ with a free variable x to be at least $\sigma(x)$: $\mathsf{size}([\![\varphi]\!]_\sigma) \ge \sigma(x)$. To ensure this we will pad formulas with $\vee\bot$. Note that if formulas are translated in a succinct manner we may not be able to prove them in the corresponding propositional proof system.*

Theorem 8. *The translation $[\![\varphi]\!]_\sigma$ is a tautology iff $\vec{x} = \sigma(\vec{x}), |\vec{X}| = \sigma(\vec{X}) \Rightarrow \varphi$ is true in the standard model.*

Proof. By induction on the structure of the formulas. □

5.2 Translating Proofs

In this section we provide a translation from proofs in the theories built over ioV^0 by adding bounded axioms – possibly in an extended language with new function symbols – to polynomial-size constant-depth H proofs with additional axioms. The sizes are measured with respect to the σ.

Definition 59 (Skolemization)**.** *Let φ be a bounded formula, Y be the outer-most existentially-quantified string variable in φ, and t be the bound on Y. The Skolemization of the variable Y of φ is obtained by replacing Y with a string function $F^Y(\vec{X}, \vec{x})$ where $(\vec{X}, \vec{x})$ is the list of free variables in the subformula starting with the quantifier $\exists Y$. The size of the Skolem function F^Y is t. The type of the value returned by Skolem function F^Y is the type of variable Y which F^Y is witnessing.*

Remark 31. *Note that Skolemization does not remove all string quantifiers.*

Example 18 (Skolemization of the comprehension axiom)**.** *Consider the φ-CA axiom:*

$$\Rightarrow \exists Y = z \ \forall x < z \ \ (x \in Y \leftrightarrow \varphi(x, A))$$

We Skolemize Y using a function $F^{\varphi\text{-}\mathsf{CA}}(z, A)$ with an output-type result and $|F^{\varphi\text{-}\mathsf{CA}}(z, A)| = z$. The resulting Skolemized axiom $\widehat{\varphi\text{-}\mathsf{CA}}$ is:

$$\Rightarrow |F^{\varphi\text{-}\mathsf{CA}}(z, A)| = z \wedge \forall x < z \ \ \Big(x \in F^{\varphi\text{-}\mathsf{CA}}(z, A) \leftrightarrow \varphi(x, A)\Big)$$

Remark 32. *The formula φ can contain string quantifiers and those quantifiers will remain intact in $\widehat{\varphi\text{-}\mathsf{CA}}$.*

Example 19 (Skolemization of the conversion axiom)**.** *Consider the $\mathsf{oiConv}_{\mathsf{str}}$ axiom:*

$$\Rightarrow \exists B = a \ \forall z < a \ \ (z \in B \leftrightarrow y + z \in X)$$

We Skolemize B using a function $F^{\mathsf{oiConv}}(a, y, X)$ with an input-type result and with size $|F^{\mathsf{oiConv}}(a, y, X)| = a$. The resulting Skolemized axiom $\widehat{\mathsf{oiConv}_{\mathsf{str}}}$ is:

$$\Rightarrow |F^{\mathsf{oiConv}}(a, y, X)| = a \wedge \forall z < a \ \ (z \in F^{\mathsf{oiConv}}(a, y, X) \leftrightarrow y + z \in X)$$

Definition 60 ($\widehat{\mathsf{ioV}^0}$)**.** *The theory $\widehat{\mathsf{ioV}^0}$ is obtained by Skolemizing the comprehension and conversion axioms of ioV^0 as given in examples 18 and 19.*

Definition 61 (Polynomially-bounded theory)**.** *A theory is* polynomially-bounded *iff all function symbols provably have polynomial bounds and all axioms are Σ^B_∞ formulas.*

Theorem 9 (Propositional Translation)**.** *Let $\mathcal{T} = \mathsf{ioV}^0 \cup \Gamma$ be a polynomially-bounded theory, $\widehat{\Gamma}$ some Skolemization of Γ, and $\varphi \in \Sigma^B_0$ a formula provable in $\mathcal{T}$. Then $[\![\varphi]\!]_\sigma$ has a polynomial-size bounded-depth H-proof from $[\![\widehat{\Gamma}]\!]_\sigma$.*

If φ is provable then so is $\varphi[\vec{x}, \vec{X}/\vec{a}, \vec{A}]$, which has the same translation. Therefore, without loss of generality, we will assume that all free variables in φ have input type. We refer to $\vec{n} = \sigma(\vec{a}, \vec{A})$ as *translation parameters* where $\vec{a}$ and $\vec{A}$ are φ's free variables.

Proof. The propositional translation has three steps:

Step 1: Skolemization lemma: If $\mathsf{ioV}^0 + \Gamma \vdash \varphi$ then $\widehat{\mathsf{ioV}^0} + \widehat{\Gamma} \vdash \varphi$.

Step 2: Translation to H plus translated axioms lemma: If $\widehat{\mathsf{ioV}^0} + \widehat{\Gamma} \vdash \varphi$ then $[\![\varphi]\!]_\sigma$ has a polynomial-size bounded-depth H-proof from $[\![\widehat{\mathsf{oiConv}}]\!]_\sigma + [\![\widehat{\mathsf{CA}}]\!]_\sigma + [\![\mathsf{Ind}]\!]_\sigma + [\![\widehat{\Gamma}]\!]_\sigma$.

Step 3: Explicit witnessing lemma: We remove the Skolem function symbols $F^{\varphi\text{-}\mathsf{CA}}$ and F^{ioConv} by providing explicit polynomial-size bounded-depth propositional formulas. We remove the translated axioms $[\![\widehat{\mathsf{oiConv}}]\!]_\sigma$, $[\![\widehat{\mathsf{CA}}]\!]_\sigma$, and $[\![\mathsf{Ind}]\!]_\sigma$ by providing explicit polynomial-size bounded-depth H proofs for them.

□

Lemma 7 (Step 1)**.** *If $\mathsf{ioV}^0 + \Gamma \vdash \varphi$, then $\widehat{\mathsf{ioV}^0} + \widehat{\Gamma} \vdash \varphi$.*

Proof of Step 1. We only need to derive the axioms of $\mathcal{T}$ in $\widehat{\mathcal{T}}$. The original axioms are derivable from the Skolemized versions using string quantifier introduction rules. □

Lemma 8 (Step 2)**.** *If $\widehat{\mathsf{ioV}^0} + \widehat{\Gamma} \vdash \varphi$, then $[\![\varphi]\!]_\sigma$ has a polynomial-size bounded-depth H-proof from $[\![\widehat{\mathsf{oiConv}}]\!]_\sigma + [\![\widehat{\mathsf{CA}}]\!]_\sigma + [\![\mathsf{Ind}]\!]_\sigma + [\![\widehat{\Gamma}]\!]_\sigma$.*

Lemma 9. *The translated axioms of* $\mathsf{io2Basic}$ *have polynomial-size bounded-depth* H*-proofs.*

Proof. We need to prove the translation of axioms of $\mathsf{io2Basic}$. The proof is similar to the proof for the translation of $\mathsf{2Basic}$ to bounded-depth Frege in [CN10, §VII.2.3, p. 168]. □

Proof of Step 2. Without loss of generality, we assume that π is a proof in free-variable free-cut free normal form. If it is not, we can first normalize it to remove any free-cuts and make sure every variable in the proof is either a free variable in the end-sequent or an eigenvariable for a $\forall\mathsf{L}$ or $\exists\mathsf{R}$ rule.

The proof π 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 axioms. $\widehat{\mathsf{ioV}^0}$ has no string quantifiers, therefore, all string quantifiers in the proof are from $\widehat{\Gamma}$.

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

For example, consider $\forall\mathsf{R}$.

$$\Gamma \Rightarrow \Delta, \forall x \leq t\ \psi \quad \longmapsto \quad [\![\Gamma]\!]_\sigma \Rightarrow [\![\Delta]\!]_\sigma, \bigwedge_{i \leq \sigma(t)} [\![x \leq t \to \psi]\!]_{\sigma[x \mapsto i]}.$$

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

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

The axioms are translated to non-logical relativized 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 $O(\mathsf{poly}(\vec{n}))$.

The translations of the axioms of $\mathsf{io2Basic}$ have polynomial-size H proofs by lemma 9. □

Remark 33. *The translated proof satisfies the following conditions:*

- $\mathsf{size}([\![\pi]\!]_{\vec{n}}) = O(\mathsf{poly}(\vec{n}))$,
- $\mathsf{ldepth}([\![\pi]\!]_{\vec{n}}) = O(1)$,
- $\mathsf{qdepth}([\![\pi]\!]_{\vec{n}}) = O(1)$.
- $depth([\![\pi]\!]_n) = depth(\pi)$, *the depth of the proof tree,*
- *all function symbols in* $[\![\pi]\!]_n$ *come from the Skolemized axioms,*

- *all string quantified cut formulas come from* $\widehat{\Gamma}$.

Lemma 10 (Step 3: φ-CA)**.** *There are polynomial-size bounded-depth propositional formulas* $\varphi^{\varphi\text{-CA},\sigma,k}$ *such that when we replace the propositional function symbols* $\alpha^{F^{\varphi\text{-CA}},\sigma,k}$ *with* $\varphi^{\varphi\text{-CA},\sigma,k}$ *the translated Skolemized axiom* $[\![\widehat{\varphi\text{-CA}}]\!]_\sigma$ *has polynomial-size bounded-depth* H*-proofs.*

Proof. Recall that the Skolemized axiom $\widehat{\varphi\text{-CA}}$ is

$$\Rightarrow |F^{\varphi\text{-CA}}(z,A)| = z \wedge \forall x < z \ \left(x \in F^{\varphi\text{-CA}}(z,A) \leftrightarrow \varphi(x,A)\right)$$

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

$$\Rightarrow [\![z = z]\!]_\sigma \wedge \bigwedge_{i\le\sigma(t)} \left([\![\varphi(x,A)]\!]_{\sigma[x\mapsto i]} \leftrightarrow [\![\varphi(x,A)]\!]_{\sigma[x\mapsto i]}\right)$$

which have $\mathsf{bdG_0}$ cut-free proofs of polynomial size. □

Lemma 11 (Step 3: φ-ioConv)**.** *There are polynomial-size bounded-depth propositional formulas* $\varphi^{\mathsf{ioConv},\sigma,k}$ *such that when we replace the propositional function symbols* $\alpha^{F^{\mathsf{ioConv}},\sigma,k}$ *with propositional formulas* $\varphi^{\mathsf{ioConv},\sigma,k}$ *the translated Skolemized axiom* $[\![\widehat{\mathsf{oiConv}}]\!]_\sigma$ *has polynomial-size bounded-depth* H*-proofs.*

Proof. Recall that the Skolemized $\widehat{\mathsf{oiConv}}$ is:

$$\Rightarrow |F^{\mathsf{oiConv}}(a,y,X)| = a \wedge \forall z < a \ (z \in F^{\mathsf{oiConv}}(a,y,X) \leftrightarrow y + z \in X)$$

We use the substring function to replace the function symbol $\alpha^{F^{\mathsf{ioConv}},\sigma,k}$ in the conversion axiom. To remove the conversion axiom, we replace $\alpha^{F^{\mathsf{ioConv}},\sigma,k}(a,y,X)$ with $([\![X[y,a]]\!]_\sigma)_k$. The axiom becomes

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

which has a $\mathsf{bdG_0}$-proof of polynomial size and bounded depth. □

Lemma 12 (Step 3: Ind)**.** *The axiom* $[\![\mathsf{Ind}]\!]_\sigma$ *also has polynomial-size bounded-depth* H *proof.*

Proof. Recall that the Ind is

$$0 \in X, \forall y < z \ (y \in X \to y + 1 \in X) \Rightarrow z \in X$$

The translation of the induction axiom becomes

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

Note that we have polynomial-size bounded-depth proofs for:

- $[\![0 \in X]\!]_\sigma \Rightarrow [\![y \in X]\!]_{\sigma[y \mapsto 0]}$
- $[\![y \leq z]\!]_{\sigma[y \mapsto i]}$ for $i \leq \sigma(z)$
- $[\![y+1 \in X]\!]_{\sigma[y \mapsto i]} \Rightarrow [\![y \in X]\!]_{\sigma[y \mapsto i+1]}$

We can combine these $\sigma(z)$ times with $[\![0 \in X]\!]_\sigma$ and

$$[\![y \leq z]\!]_{\sigma[y \mapsto i]}, [\![y \in X]\!]_{\sigma[y \mapsto i]} \Rightarrow [\![y+1 \in X]\!]_{\sigma[y \mapsto i]}$$

with $i \leq \sigma(z)$ using Cut and $\wedge$R to obtain $[\![z \in X]\!]_\sigma$.

□

Remark 34 (Computability of the translation)**.** *With the exception of the normalization in Step 2 the translation of a proof is computable in polynomial time. Normalization requires the elimination of free cuts which in general is not computable in polynomial time. As mentioned the size of the translated proof is polynomial. However, the polynomial bounding the size of the resulting proof depends on the terms inside the proof and therefore the translation is not* uniformly *polynomial-time computable even when restricted to proofs in normal form. E.g. consider the formula $\exists X = t\ \varphi$. We need t existentially quantified propositional variables to translate this* NP *predicate. This can be avoided if we use a more succinct encoding for quantified formulas. If we further restrict the size of the bounding terms in the proof to $O(n^k)$ for some fixed k, then the translation is uniformly computable in polynomial time.*

Next, we prove a lemma about the bounds on input-type variables in a proof which will be used in chapter 8.

Lemma 13 (Linear bounds)**.** *Let π be a free-variable free-cut free normal proof of formula φ in an io-theory $\mathcal{T}$. Assume that all input-type quantifiers in $\mathcal{T}$ and φ are bounded by input-type terms. Then all input-type variables in π can be bounded by input-type terms in the parameters of the proof.*

Proof. Assume that the proof is in free-cut free free-variable normal form. Therefore, each formula in the proof is either a subformula of the end-sequent or an axiom, and each free variables in the proof is either an eigenvariable or a parameter.

An easy argument shows that if a variable is quantified in some formula in the proof it is bounded by a linear term in the free variables of that formula because the formula is either a subformula of an axiom or the end-sequent. Therefore, we only need to consider variables which are not quantified anywhere in the proof. But variables which are not quantified in the proof are the parameters of the end-sequent and bound themselves. □

5.3 Extending Propositional Translation to ioVC

We have shown how to remove the translated axioms of ioV^0 from the translated proofs. However, the translations of the axioms of Γ are still part of the translated proofs. To remove these remaining axioms from the translated proofs we will follow a similar strategy:

- Find suitable formulas to substitute for the propositional function symbols.
- Provide propositional proofs for the resulting axioms.

Chapter 6

From Nonuniform to Uniform: Soundness

In chapter 5 we provided a propositional translation that maps io-typed theories to propositional proof complexity classes. The propositional translation provides an upper bound on the propositional proof complexity class corresponding to a theory. In this chapter we will provide a kind of converse by using soundness tautologies for proof complexity classes defined in section 4.5. The propositional proof complexity system corresponding to a theory is minimal:

- The theory proves the soundness of the corresponding proof complexity class.
- The corresponding proof complexity class polynomially simulates any proof complexity class that the theory proves its soundness.

6.1 Maximal Proof Complexity Class Corresponding of Theory

We will use $\mathsf{Snd}(\mathcal{F})$ for both first-order formulas and their propositional translations expressing the soundness of $\mathcal{F}$. Note that section 4.5 can be taken to be the propositional translation of first-order soundness formulas.

Theorem 10. *Let $\mathcal{T}$ be a theory extending ioV^0. Let $\mathcal{F}'$ be a nice proof complexity class whose soundness is provable in $\mathcal{T}$. Let $\mathcal{F}$ be a proof complexity class which contains $\mathcal{R}$ (polynomial-size bounded-depth Frege) and is closed under substitution and proves the translation of Σ^B_0 theorems of $\mathcal{T}$. Then $\mathcal{F}$ weakly contains $\mathcal{F}'$, i.e. for every $\{\varphi_n\}_n$ if $\mathcal{F}' \vdash \{\varphi_n\}_n$ then $\mathcal{F} \vdash \{\text{“}\vDash \varphi_n\text{”}\}_n$.*

Proof. It follows from $\mathcal{T} \vdash \mathsf{Snd}(\mathcal{F}')$ and $\mathsf{Snd}(\mathcal{F}')$ being Σ_0^B that $\mathcal{F} \vdash \mathsf{Snd}(\mathcal{F}')$. $\mathcal{F}$ is closed under substitution; therefore, $\mathcal{F} \vdash \overline{\mathsf{Snd}(\mathcal{F})}$. It follows from theorem 7 that $\mathcal{F}$ weakly contains $\mathcal{F}'$. □

6.2 Evaluation of Bounded-depth Formulas in $\mathsf{io}\mathsf{V}^0$

Theorem 11. *For every d there is a Σ_0^B formula $\vDash_d$ expressing the evaluation of depth d formulas such that $\mathsf{io}\mathsf{V}^0$ proves $\vDash_d$ respects the structure of the formulas, i.e. $\mathsf{io}\mathsf{V}^0$ proves Tarski's inductive definition for $\vDash_d$.*

Proof. The existence of the formula follows from the fact that we can evaluate depth d formulas in AC^0 and therefore can express it as a Σ_0^B formula. We need to show that it respects the inductive definition of truth, e.g. if Z and Z_is are depth d formulas and Z is the conjunction of Z_is for $0 \leq i < k$, then Z is true iff for all $i < k$, Z_i is true.

We use the connection language for the encoding of circuits. Let Z be a string that encodes a depth d formula with m gates on n inputs. We represent a formula as a labeled dag. We refer to the gates in circuits by numbers. We encode the input gate number i as the number i. We write $i \to j$ when gate i is a child of gate j. Let $g_i \in \{\ulcorner\wedge\urcorner, \ulcorner\vee\urcorner, \ulcorner\neg\urcorner, \ulcorner\bot\urcorner, \ulcorner\top\urcorner, 0, \ldots, n-1\}$ be the type of gate i. Let I be a string encoding the input for a formula. Note that we can express that the depth of a gate is at most d by stating that there is no path of length $d+1$ in the dag:

$$\mathsf{depth}_d(Z, i) := \forall j_0, \ldots, j_d \ \neg(j_0 \to j_1 \wedge j_1 \to j_2 \wedge \cdots \wedge j_{d-1} \to j_d \wedge j_d \to i).$$

We define the formula $\vDash_d$ inductively. We $I \vDash_d Z, i$ if gate i is true in the formula encoded by Z on input given by I. In the base case where $d = 0$ we only have $\bot$, $\top$, and input gates. We can define $I \vDash_0 Z, i$ as

$$g_i = \ulcorner\top\urcorner \vee (0 \leq g_i < n \wedge g_i \in I).$$

For depth $d+1$ we define $I \vDash_{d+1} Z, i$ as

$$(\neg\mathsf{depth}_d(Z, i) \wedge g_i = \ulcorner\wedge\urcorner \wedge \forall j \to i \ (I \vDash_d Z, j)) \vee$$

$$(\neg\mathsf{depth}_d(Z, i) \wedge g_i = \ulcorner\vee\urcorner \wedge \exists j \to i \ (I \vDash_d Z, j)) \vee$$

$$(\neg\mathsf{depth}_d(Z, i) \wedge g_i = \ulcorner\neg\urcorner \wedge \exists! j \to i \ \neg(I \vDash_d Z, j)) \vee$$

$$(\mathsf{depth}_d(Z, i) \wedge (I \vDash_d Z, i)).$$

Now $I \models_d Z$ can be defined as $I \models_d Z, 0$. Proving that $\models_d$ respects the inductive definition of the truth is straightforward. □

Remark 35. *Note that $\models_d$ does not need to check that the input really encodes a formula, i.e. this is a promise problem.*

6.3 ioV^0 Proves the Soundness of Bounded-depth Frege

Theorem 12. ioV^0 *proves the soundness of polynomial-size cut-free* Frege.

Proof. We argue inside ioV^0 using the fact that we can compute AC^0 functions and use induction on AC^0 properties of objects. Let π be a cut-free Frege proof of a propositional formula φ of depth d with free variables $\vec{p}$. We put the proof in free-variable normal form by replacing any propositional variable which is not free in φ with $\bot$. It is easy to check that this is still a valid proof.

We can prove the subformula property for the rules. Since there are no cuts in the proof every formula in the proof is a subformula of φ. Therefore, all formulas have depth $\leq d$.

Let τ be an arbitrary assignment to φ. We want to show that $\tau \models_d \varphi$. We do so by induction on the truth of sequents in the proof. It is easy to verify the truth of axioms and structural rules. For the logical connective introduction rules, we use the fact that truth respects the structure of the formulas. □

Theorem 13. ioV^0 *proves the soundness of bounded-depth* Frege *proofs.*

Proof. The proof is similar to the proof of theorem 11. The only difference is that we now have the cut rule over depth $\leq d$ formulas. The subformula property holds: every formula is either a subformula of φ or a cut formula. Since all cut formulas have depth $\leq d$ every formula in the proof has depth $\leq d$.

We only need to check the induction step for the cut rule for depth $\leq d$ formulas.

$$\frac{\Gamma \Rightarrow \Delta, \psi \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ \textbf{Cut}$$

The truth assignment τ satisfies the top sequents in the cut rule and ψ is a depth d formula. We have to show that τ satisfies $\Gamma \Rightarrow \Delta$. This is straightforward case analysis: either $\tau \models_d \psi$ or $\tau \not\models_d \psi$. In the first case, the truth of top right sequent implies that $\Gamma \Rightarrow \Delta$ is true. In the second case, the truth of top left sequent implies that $\Gamma \Rightarrow \Delta$ is true. □

Chapter 7

Nondeterministic Time Space and Alternating Time

In this chapter, we prove that functions computable by nondeterministic Turing machines in polynomial time and fractional space ($\mathsf{NTimeSpace}(n^{O(1)}, n^{\varepsilon})$) can be computed by alternating Turing machines in fractional time with constant number of alternations ($\mathsf{AltTime}(O(1), O(n^{\varepsilon}))$), which we view as a uniform version of unbounded fan-in propositional circuit families of subexponential size and bounded depth ($\mathsf{DepthSize}(O(1), 2^{n^{\varepsilon}})$). We use a formalization of this result in chapter 8 about our theory n^{ε}-ioV^{∞} for $\mathsf{AltTime}(O(1), O(n^{\varepsilon}))$.

The result is based on the general theme of trading time for alternations when space is small [FLM+05; AK10; Gol12] and goes back to Nepomnjascij's theorem [Nep70].

Theorem 14 (Nepomnjascij's theorem)**.** *For all $k > 0$ and $\varepsilon < 1$:*

$$\mathsf{NTimeSpace}(n^k, n^{\varepsilon}) \subseteq \mathsf{AltTime}(O(1), O(n)) = \mathsf{LTH} = \Sigma_{\infty}^{B \leq n}$$

where LTH *stands for* Linear Time Hierarchy*.*

Note that by alternating Turing machines we mean the modified version of alternating Turing machines used in the literature [Ruz79; Imm99; Vol99; CN10] for small classes where the input tape is read only and has both ends marked by a special symbol and there is a special tape for writing the index of input bits and reading them like a random access memory.

Let d be a natural number and $t(n) \in \Omega(\lg n)$ be a term. Let $\Sigma_d^{B \leq t(n)}$ be the class of first-order formulas with at most d alternations of string quantifiers with bound $O(t(n))$ where n is the size of its free variables. Similarly, $\Sigma_d^{q \leq t(n)}$ is the class of quantified

propositional formula families of polynomial size with at most d quantifier alternations (including the AND and OR gates) and each propositional quantifier quantifies over at most $O(t(n))$ propositional variables. We use $d = \infty$ for the union of these classes over all $d \in \mathbb{N}$.

Note that [Ruz79; Imm99; CN10]

$$\Sigma_\infty^{B \le \lg n} = \Sigma_0^B = \mathsf{FO} = \mathsf{LH} = \mathsf{AltTime}(O(1), O(\lg n)) = \mathsf{AC}^0$$

and

$$\Sigma_\infty^{B \le n^{O(1)}} = \Sigma_\infty^B = \mathsf{SO} = \mathsf{PH} = \mathsf{AltTime}(O(1), n^{O(1)}) = \mathsf{DLogTime\text{-}uniform}\ \Sigma_\infty^q.$$

More generally it is not difficult to see the following holds for constructable $t(n) \in \Omega(\lg n)$:

Theorem 15. $\Sigma_\infty^{B \le t(n)} = \mathsf{AltTime}(O(1), O(t(n))) = \mathsf{DLogTime\text{-}uniform}\ \Sigma_\infty^{q \le t(n)}$.

Proof. The proof is similar to the proof of $\mathsf{FO} = \mathsf{AltTime}(O(1), O(\lg n)) = \mathsf{AC}^0$. See [Imm99, Theorem 5.30] and [Vol99, Theorem 4.69 and Theorem 4.73]. □

Remark 36. *The class* $\mathsf{NC}^k = \mathsf{AltSpace}(O(\lg^k n), O(\lg n))$ *is generalized in [BCP83] to define the class* $\mathsf{NC}^k(R(n)) = \mathsf{AltSpace}(O(\lg^k R(n)), O(\lg R(n)))$. *E.g.* $\mathsf{NC}(2^{n^{O(1)}}) = \mathsf{AltSpace}(n^{O(1)}, n^{O(1)}) = \mathsf{PSpace}$.

In a similar way we can generalize the class $\mathsf{AC}^0 = \mathsf{AltTime}(O(1), O(\lg n))$ *to define the class* $AltTime(O(1), O(\lg R(n)))$. *Using this definition with* $R(n) = 2^{t(n)}$ *we obtain* $\mathsf{AC}^0(2^{t(n)}) = \mathsf{AltTime}(O(1), O(t(n)))$, *an alternative argument through for the class* $\mathsf{AltTime}(O(1), O(t(n)))$ *with* $t(n) = n^\varepsilon$ *being a uniform version of subexponential-size bounded-depth circuits.*

Theorem 16. $\Sigma_\infty^{q \le t(n)} \subseteq \mathsf{DepthSize}(O(1), 2^{O(t(n))})$.

Proof. We only need to remove the propositional quantifiers. We can do so by replacing $\exists$ and $\forall$ quantifiers with OR and AND gates of fan-in $2^{t(n)}$. The resulting propositional circuits have a constant depth. Let the size of the original circuit be $s \in n^{O(1)}$. The size of the resulting circuit is $s2^{O(t(n))} = 2^{O(t(n)) + \lg s} = 2^{O(t(n))}$ since $t(n) \in \Omega(\lg n) \subseteq \Omega(\lg s)$. □

Therefore, in the rest of this chapter we will work mainly with $\Sigma_\infty^{B \le t(n)}$ formulas as a uniform version of subexponential-size bounded-depth circuits.

Theorem 17 (Main theorem). *Nondeterministic polynomial-time $n^{o(1)}$-space functions can be computed by subexponential-size constant-depth circuits. More formally, for all $\delta > 0$ we have*

$$\begin{aligned}\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)}) &\subseteq \Sigma_{O(1/\delta)}^{B\le n^\delta} \\ &= \mathsf{AltTime}(O(1/\delta), O(n^\delta)) \\ &\subseteq \mathsf{DepthSize}(O(1/\delta), 2^{O(n^\delta)}).\end{aligned}$$

Compared to Nepomnjascij's theorem the class $\mathsf{AltTime}(O(1/\delta), O(n^\delta))$ with $\delta < 1$ is sublinear-time and is contained in $\mathsf{AltTime}(O(1), O(n))$ of Nepomnjascij's theorem. Moreover, the class $\mathsf{AltTime}(O(1/\delta), O(n^\delta))$ with $\delta < 1$ corresponds to uniform subexponential-size bounded-depth circuits as discussed above.

Theorem 17 with the dependence between constants is explicitly stated is as follows:

Theorem 18 (Restatement of theorem 17)**.** *For all k, $\varepsilon(n) = o(1)$, and $\delta > 0$ we have*

$$\mathsf{NTimeSpace}(n^k, n^\varepsilon) \subseteq \Sigma_{\frac{4k}{\delta}+1}^{B\le n^\delta}.$$

We obtain theorem 18 as a corollary of the following more general simulation lemma. In the following, $t(n) = O(n^k)$ is the original time complexity of the algorithm, $s(n) = O(n^\varepsilon)$ is the original space complexity of the algorithm, the constant d is essentially the string alternation depth of the formula (more exactly, it is the number of layers in the layered brute-force technique [AK10]), and string quantifiers are bounded by $O(n^\delta)$ where $0 < \delta$ can be chosen arbitrarily.

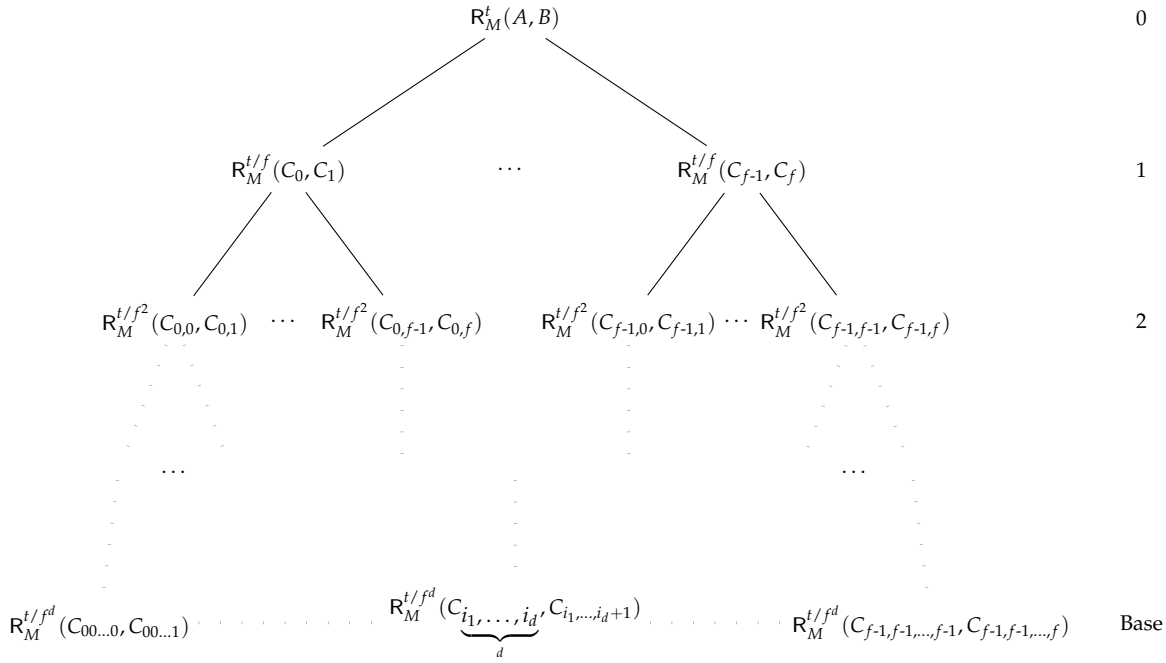
Lemma 14. *For all positive $k, \varepsilon(n), d, \delta$ such that $(k + \varepsilon - \delta/2)/d + \varepsilon + \lg d/\lg n \le \delta/2$ we have*

$$\mathsf{NTimeSpace}(n^k, n^\varepsilon) \subseteq \Sigma_{2d+1}^{B\le n^\delta}.$$

Assume that we want to verify the output of a machine M in $\mathsf{NTimeSpace}(t, s)$. Each configuration of M has size $O(s)$. Therefore, the computation of M has size $O(st)$. We use divide and conquer to guess and verify the computation. We divide the computation into f blocks and check the computation in each block. We repeat this recursively for d levels. This gives an f-ary tree of depth d. The size of each computation block in the last level is $O(st/f^d)$ which we guess and verify directly in $\mathsf{NTime}(O(st/f^d))$. Along each path in the recursive division, we have to guess fd starting and ending configurations (one for each block along the path). See figure 7.1.

In total we guess fsd bits along each path in the tree. The resulting algorithm is in

$$\mathsf{AltTime}(2d + 1, O(st/f^d + fsd))$$



In this figure, $\mathsf{R}^{t}_{M}(A,B)$ denotes the fact that configuration B can be reached from configuration A in t steps. We use divide and conquer to break down the computation into smaller pieces that can be guessed and verified independently through the following procedure which is repeated d times:

$$\mathsf{R}^{t}_{M}(A,B) :\equiv \exists C_0,\ldots,C_f = s \ \left(C_0 = A \wedge C_f = B \wedge \forall i < f \ (\mathsf{R}^{t/f}_{M}(C_i,C_{i+1}))\right)$$

Figure 7.1: Divide and Conquer Algorithm for Configuration Reachability

This gives two competing terms containing d: fsd and st/f^d. To obtain our target result we have to make both of these terms smaller than $O(n^\delta)$. The conditions stated in lemma 14 reflect these requirements.

We first prove the following brute-force theorem that we will later use for the last step in our proof of lemma 14 where st will be $o(n)$:

Lemma 15. $\mathsf{NTimeSpace}(t,s) \subseteq \Sigma_1^{B\leq st}$

Proof. We have to provide a $\Sigma_1^B(st)$ formula $\mathsf{R}_M^{t,s}(A,B)$ expressing that configuration B can be reached from configuration A in t steps, a Σ_0^B formula $\mathsf{Accept}(B)$ which expresses that B is an accepting configuration, and a Σ_0^B formula $\mathsf{Init}(A,X)$ which expresses that A is an initial configuration with input x. The resulting formula is

$$\exists A,B = s \ \left(\mathsf{Init}(A,X) \wedge \mathsf{Accept}(B) \wedge \mathsf{R}_M^{t,s}(A,B)\right).$$

The latter two conjuncts are standard FO functions and therefore expressible in Σ_0^B. For $\mathsf{R}_M^{t,s}(A,B)$ we use the following formula:

$$\exists C = s(t+1) \ \ (C[0,s] = A \wedge C[ts,s] = B \wedge \forall i < t \ \forall j < s$$

$$\mathsf{Trans}_M(C[is+j-1], C[is+j], C[is+j+1], C[(i+1)s+j]))$$

where Trans_M expresses the finite function which verifies that jth symbol of ith configuration $C[is+j]$ has been computed correctly from the previous configuration (as in Cook-Levin theorem). □

Now we can continue with the proof of lemma 14.

Proof of lemma 14. Note that we can reduce the running time by a factor of f by using two quantifiers: guess f intermediate configurations and check that each follows from the previous one.

Let $A \vdash_M^m B$ mean that M can reach configuration B starting from configuration A in m steps. Then $A \vdash_M^m B$ is equivalent to

$$\exists C = (f+1)s \ \left(C[0,s] = A \wedge C[fs,s] = B \wedge \forall i < f \ \ C[is,s] \vdash_M^{\lceil m/f \rceil} C[(i+1)s,s]\right)$$

Repeating this trade-off of time for alternation d times on $A \vdash_M^t B$, we end up with $2d$ quantifier blocks ($2d$ alternation) followed by $C[is,s] \vdash_M^{\lceil t/f^d \rceil} C[(i+1)s,s]$ and the last part is in $\Sigma_1^{B\leq st/f^d}$ by lemma 15. This shows that $\mathsf{Init}(X) \vdash_M^t \mathsf{Accept}$ is in $\Sigma_{2d+1}^{B\leq st/f^d+fsd}$.

Next, we have to fix the values of f and d. Note that we want to get the bounds to be $O(n^\delta)$. Let $f = n^b$, $s = n^\varepsilon$, and $t = n^k$. Then we obtain the result since we assumed $fsd = n^b n^\varepsilon d = dn^{\varepsilon+b} \leq n^{\delta/2}$ and $st/f^d = n^\varepsilon n^k / n^{bd} = n^{\varepsilon+k-bd} \leq n^{\delta/2}$. □

Now we derive theorem 18 from lemma 14.

Proof of theorem 18. Let $L \in \mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)})$. Let $\varepsilon = o(1)$ and constant k such that $L \in \mathsf{NTimeSpace}(n^k, n^\varepsilon)$. We want to show that $L \in \Sigma^{B \leq n^\delta}_{O(1/\delta)}$. Let $\delta > 0$ be arbitrary. We will fix the values of other parameters such that the conditions for lemma 14 are satisfied. Because of choice of $b = (k + \varepsilon - \delta/2)/d$ we only need to make sure that

$$b + \varepsilon + \lg d / \lg n \leq \delta/2$$

The term $\lg d / \lg n$ is $o(1)$ and since we only care about large enough n we can drop it and use a strict inequality. We need to show that

$$b + \varepsilon < \delta/2$$

Replacing the value of b and rearranging the terms we obtain

$$k/(d+1) < \delta/2 - \varepsilon$$

But $\varepsilon = o(1)$ so we only need to show

$$k/(d+1) < \delta/2$$

Equivalently

$$2k/\delta < d + 1$$

So we can take $d = 2k/\delta$. Note that $d = O(1/\delta)$. □

As a corollary of theorem 17 we prove that NL has subexponential-size bounded-depth circuits [Gol12].

Corollary 2. NL *and* SC *have subexponential-size bounded-depth circuits.*

Proof. Since $\mathsf{NL} = \mathsf{NTimeSpace}(n^{O(1)}, \lg n)$ and $\mathsf{SC} = \mathsf{TimeSpace}(n^{O(1)}, \lg^{O(1)} n)$ by theorem 17 we have

$$\begin{aligned}
\mathsf{NL}, \mathsf{SC} &\subseteq \mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)}) \\
&\subseteq \Sigma_{O(1/\delta)}^{B \leq n^{\delta}} && \text{(theorem 17)} \\
&= \mathsf{AltTime}(O(1/\delta), O(n^{\delta})) \\
&\subseteq \mathsf{DepthSize}(O(1/\delta), 2^{O(n^{\delta})}).
\end{aligned}$$

□

Chapter 8

Theories for Uniform Subexponential Bounded-depth Circuit Families and NC^1

In this chapter we define io-typed theories $t(n)$-ioV^∞ for $\mathsf{AltTime}(O(1), O(t(n)))$ for $t(n) = n^\varepsilon$ with $\varepsilon < 1$ (section 8.1) and ioVNC^1 for NC^1 (section 8.2). We show that the former contains the latter by formalizing the complexity class containment $\mathsf{NC}^1 \subseteq \mathsf{AltTime}(\mathsf{O}(1), \mathsf{O}(\mathsf{t}(\mathsf{n})))$ of chapter 7 inside $t(n)$-ioV^∞ (section 8.3, theorem 26). This provides a uniform version of the result in [FPS15] that polynomial-size Frege proofs can be translated to subexponential-size bounded-depth Frege proofs.

Finally, we derive the nonuniform result as a corollary of our uniform result (section 8.4, corollary 7) by combining $t(n)$-$\mathsf{ioV}^\infty \vdash \mathsf{ioVNC}^1$ (theorem 26), $\mathsf{ioVNC}^1 \vdash \mathsf{Snd}(\mathsf{Frege})$ (theorem 25), the universality of $\mathsf{Snd}(\mathsf{Frege})$ for polynomial-size Frege (theorem 7), and the propositional translation from $t(n)$-ioV^∞ to polynomial-size $t(n)$-bdG_∞ and subexponential-size bounded-depth Frege (theorem 20).

Convention 6 ($t(n) = n^\varepsilon$)**.** *Throughout this chapter, $t(n) = n^\varepsilon$ where $\varepsilon = \frac{1}{d} < 1$ for some fixed d.*

8.1 Theory $t(n)$-ioV^∞ for Subexponential-size Bounded-depth Frege

In this section we introduce an io-typed theory that corresponds to the class of functions computable by uniform subexponential-size bounded-depth circuits. As discussed in chapter 7 we consider $\mathsf{AltTime}(O(1), O(t(n)))$ as our uniform version of

subexponential-size bounded-depth circuits $\mathsf{DepthSize}(O(1), 2^{O(t(n))})$: the circuit evaluation problem for circuit families in $\mathsf{DepthSize}(O(1), 2^{O(t(n))})$ where the circuit is given using a random access tape can be computed in $\mathsf{AltTime}(O(1), O(t(n)))$, and $\mathsf{AltTime}(O(1), O(t(n))) \subseteq \mathsf{DepthSize}(O(1), 2^{O(t(n))})$.

We use the following subclass of bounded formulas.

Definition 62 ($\Sigma_\infty^{B\leq t(n)}$)**.** *We call a formula $\Sigma_\infty^{B\leq t(\vec{n})}$ 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 $\Sigma_\infty^{B\leq t(n)}$ in place of $\Sigma_\infty^{B\leq t(\vec{n})}$. In such cases, n can be considered to be the total size of the free variables.*

The class of $\Sigma_\infty^{B\leq t(n)}$ formulas captures exactly the functions which are computable in $\mathsf{AltTime}(O(1), O(t(n)))$ by theorem 15 in chapter 7. Therefore, we use a comprehension axiom for $\Sigma_\infty^{B\leq t(n)}$ formulas to define the theory, as we used Σ_0^B comprehension for defining our theory for AC^0.

8.1.1 Theory $t(n)$-ioV^∞

We define the theory $t(n)$-ioV^∞ for the complexity class $\mathsf{AltTime}(O(1), O(t(n)))$ by adding the comprehension axiom $\Sigma_\infty^{B\leq t(n)}$-CA to provide the necessary computational power. Note that $t(n)$ is a term of number sort which bounds the quantified string variables in φ and only contains input-type variables. 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. The language is extended to contain the new function and the theory contains the defining axiom of the function. We include $t(n) = n^{\frac{1}{d}}$ from definition 14. We define our theory $t(n)$-ioV^∞ as follows:

Definition 63. *The io-typed theory $t(n)$-ioV^∞ is defined as*

- $t(n)$-$\mathsf{ioV}^\infty := \mathsf{ioV}^0 + \Sigma_\infty^{B\leq t(n)}$-CA.

Remark 37. *It is not difficult to show that ioV^0 is equivalent to $\lg n$-ioV^∞. Using Σ_0^B-CA and definability of Bit in ioV^0 we can prove $\Sigma_\infty^{B\leq \lg n} = \Sigma_0^B$ in ioV^0. The function $\lg x$ and the relation $\mathsf{Bit}(x, y)$ are definable in ioV^0 (see [CN10, §III.3.3] and [Imm99, §1.2.1]). Using comprehension on $\mathsf{Bit}(x, a)$ and $\lg a$ we can convert unary numbers to binary numbers of logarithmic size in ioV^0. Similarly, using comprehension and $|.|$ we can convert a binary number of size $\lg a$ to a unary number of size a.*

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

Theorem 19 (Provably Total Functions of $t(n)$-ioV^∞)**.** *The provably total functions of the theory $t(n)$-ioV^∞ are exactly functions computable in $\mathsf{AltTime}(O(1), O(t(n)))$ where n is the size of the arguments.*

Proof. 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 $\Sigma_\infty^{B\leq t(n)}$. On the other hand, by witnessing every provably total function of $t(n)$-ioV^∞ belongs to $\mathsf{AltTime}(O(1), O(t(n)))$. □

8.1.2 Propositional Translation

Next, we discuss the propositional translation of proofs in $t(n)$-ioV^∞ to polynomial-size proof families in $t(n)$-bdG_∞ and from there to subexponential-size bounded-depth G_0 proofs.

Definition 64 ($\Sigma_\infty^{q\leq t(n)}$)**.** *Let $\Sigma_\infty^{q\leq t(n)}$ denote the class of those Σ_∞^q quantified propositional formula families with a bounded number of alternations (including AND and OR gates) where the number of quantified propositional variables is bounded by $O(t(n))$.*

Definition 65 ($t(m)$-bdG_∞)**.** *The proof class $t(m)$-bdG_∞ is the subclass of bdG_∞ proofs where cuts are restricted to $\Sigma_\infty^{q\leq t(m)}$ formulas where m is the size of the end-sequent. In addition, the total number of eigenvariables in each sequent of $t(m)$-bdG_∞ proofs must be $O(t(m))$.*

Note that the result of propositional translation of a $\Sigma_\infty^{B\leq t(n)}$ formula is a $\Sigma_\infty^{q\leq t(n)}$ formula family.

Theorem 20 (Propositional Translation)**.** *If $\varphi \in \Sigma_0^B$ is provable in $t(n)$-ioV^∞, then $\{[\![\varphi]\!]_{\vec{n}}\}_{\vec{n}}$ has polynomial-size $t(n)$-bdG_∞ proofs and subexponential-size bounded-depth G_0 proofs.*

Proof. The proof follows the argument in section 5.2. We first Skolemize the comprehension axiom. Given a proof in the Skolemized theory, we translate it to a proof family in H with translated axioms. Note that we only have a fixed number of axioms in the translated proofs. We remove the axioms of ioV^0 as discussed step 3. The only remaining axioms in the translated proof are the translations of the Skolemized $\Sigma_\infty^{B\leq t(n)}$-CA axioms, which can be removed as the translations of Σ_0^B-CA axioms were removed. The result is a proof in H with no axioms or function symbols.

Furthermore, the only quantified propositional variables in the proof are from the translated $\Sigma_\infty^{B\leq t(n)}$ formulas in the translated $\Sigma_\infty^{B\leq t(n)}$-CA axioms. By lemma 13 the input-type terms are bounded by linear terms in parameters. The free variables in the

$\Sigma_\infty^{B\leq t(n)}$-CA axioms were of input-type. Therefore, the bounding terms for quantifiers are $O(t(n))$. As a result, we have a polynomial-size $t(n)$-bdG$_\infty$ proof family.

Since there are only a fixed number of axioms there are at most $O(t(n))$ quantified propositional variables. To obtain a subexponential-size bounded-depth G$_0$ proof we simply replace existential and universal propositional quantifiers with $\bigwedge$ and $\bigvee$ of possible values for those variables and quantifier introduction rules with $\bigwedge$ and $\bigvee$ introduction rules. □

8.1.3 Soundness

First, note that as a corollary of theorem 13 we have

Corollary 3. *The soundness of* bdFrege *is provable in* $t(n)$-ioV$^\infty$.

Theorem 21. *The soundness of* $t(n)$-bdG$_\infty$ *is provable in* $t(n)$-ioV$^\infty$.

Proof. The argument follows the same structure of provably of soundness results. By the subformula property, all formulas in the proof are $\Sigma_\infty^{q\leq t(n)}$ whose truth can be formalized in $t(n)$-ioV$^\infty$. The rest follows from induction on the truth of sequents in the proof on a given truth assignment for parameters of the proof. □

8.2 Theory ioVNC^1 and Polynomial-size Frege proofs

We define an io-typed version of the theory VNC^1 from [CN10]. As discussed in section 3.4, since we do not have the usual composition we need a more powerful comprehension axiom. Note that the monotone balanced Boolean formula evaluation problem MBBFE is complete for NC^1 under AC^0 reductions [CN10, Theorem IX.5.2].

8.2.1 Theory ioVNC^1

Let us encode an MBBFE instance for a formula φ as a string Z of size $2s$ where the first half of Z encodes the formula tree and the second half of Z encodes the inputs. A string Y of size s is a computation of Z if it assigns correct values to the gates of Z.

Definition 66. *The formula* MBBFE-Comp(Y, Z) *states that* Y *is a computation of* Z *and is defined as:*

- $\mathsf{Comp}(Y, Z) := \forall z < |Y|$
  $[(z \in Z \to (z \in Y \leftrightarrow 2z \in Y \wedge 2z+1 \in Y)) \wedge$
  $(z \notin Z \to (z \in Y \leftrightarrow 2z \in Y \vee 2z+1 \in Y))]$

Definition 67. *The comprehension axiom for the Σ_0^B closure of* MBBFE *($\Sigma_0^B(\mathsf{MBBFE})$) denoted by $\Sigma_0^B(\mathsf{MBBFE})$-CA is defined as:*

- $\Sigma_0^B(\mathsf{MBBFE})\text{-CA} := \exists Y = s\ \exists Z = 2s\ \ [\forall x < 2s\ \ (x \in Z \leftrightarrow \varphi(x, A)) \wedge \mathsf{Comp}(Y, Z)]$ *where $\varphi \in \Sigma_0^B$.*

Definition 68. *The theory* ioVNC^1 *is defined as:*

- $\mathsf{ioVNC}^1 := \mathsf{ioV}^0 + \Sigma_0^B(\mathsf{MBBFE})\text{-CA}$

The theory ioVNC^1 corresponds to NC^1. We have the following theorem about the $\exists^B\Sigma_0^B$ definable functions:

Theorem 22. *The provably total functions of* ioVNC^1 *are precisely* NC^1 *functions.*

Proof. The proof is similar to the proof for VNC^1 in [CN10]. For one direction we note that the witnessing theorem still applies. Model theoretically, we can show that any model of VNC^1 can be viewed as a model of ioVNC^1 by interpreting input and output types the same way.

For the other direction, consider any NC^1 function. It can be obtained by composing MBBFE with an AC^0 function. Let φ be a Σ_0^B formula that represents the AC^0 function. We can define the NC^1 function using a $\exists^B\Sigma_0^B$ formula as $\Sigma_0^B(\mathsf{MBBFE})$ with this φ (without the leftmost Y quantifier for the computation). This formula defines the NC^1 function. By the comprehension axiom for $\Sigma_0^B(\mathsf{MBBFE})$ the function is provably total. □

8.2.2 Propositional Translation

We can translate a proof in ioVNC^1 to a family of polynomial-size Frege proofs. This can be achieved directly using the method in chapter 5. After translation we need to provide explicit formulas witnessing $\Sigma_0^B(\mathsf{MBBFE})$ and prove the result of this replacement on the comprehension axiom.

Remark 38. *Alternatively, we can use the fact that any provable theorem of our theory is also provable in* VNC^1 *and use the translation for* VNC^1 *from [CN10].*

8.2.3 Formalizing Truth for Boolean Formulas

It is easy to define truth for balanced Boolean formulas in ioVNC^1. We need to balance formulas and provably so in ioVNC^1 to define truth for unbalanced Boolean formulas. It

turns out that ioVNC^1 can prove Buss's result [Bus87; Bus93] that (unbalanced) Boolean formulas can be evaluated in ALogTime (which is equivalent to uniform NC^1). Buss's proof [Bus93] is formalized in VNC^1, see [CN10, pp. 410-424]. We can check that the argument in VNC^1 only uses intermediate values of linear size.

Theorem 23. ioVNC^1 *proves the totality and correctness of Buss's* ALogTime *algorithm [Bus93] 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 does not need to encode the values obtained for the 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 Σ_0^B(MBBFE)-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 function 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 do not need to compute any global function of output-type objects. So the argument in [CN10] still works. □

Corollary 4. *The (unbalanced) Boolean formula evaluation is provably total in* ioVNC^1.

Corollary 5. *There are* $\exists^B\Sigma_0^B$ *and* $\forall^B\Sigma_0^B$ *formulas provably equivalent in* ioVNC^1 *formalizing the satisfaction relation* $\vDash$ *for propositional formulas respecting its inductive definition.*

8.2.4 Proving Soundness

We can prove the soundness of polynomial-size Frege proofs in ioVNC^1. 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 24. $\mathsf{ioVNC}^1 \vdash \mathsf{Snd}(\mathsf{BalancedFrege})$.

Using the formalization of truth for unbalanced formulas in the previous section we can prove the soundness of (unbalanced) Frege in ioVNC^1.

Theorem 25. $\mathsf{ioVNC}^1 \vdash \mathsf{Snd}(\mathsf{Frege})$.

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 straightforward. For the induction step, we have to show that the rules preserve the truth of sequents. The correctness of all rules can be verified by case analysis as before. □

8.3 $\mathsf{ioVNC}^1 \subseteq t(n)\text{-}\mathsf{ioV}^\infty$

In this section we prove that $t(n)\text{-}\mathsf{ioV}^\infty \vdash \mathsf{ioVNC}^1$ where $t(n) = n^\varepsilon$ by providing and proving the correctness of an $\mathsf{AltTime}(O(1), O(t(n)))$ algorithm for $\Sigma_0^B(\mathsf{MBBFE})$. This is essentially formalizing the argument for $\mathsf{NC}^1 \subseteq \mathsf{AltTime}(\mathsf{O}(1), \mathsf{O}(\mathsf{t}(\mathsf{n})))$ in chapter 7.

Remark 39. *The following results about* NC^1 *can be generalized to other nice uniform complexity classes inside* $\mathsf{NTimeSpace}(n^{O(1)}, n^{o(1)})$ *like* NL.

Theorem 26. *The theories* $t(n)\text{-}\mathsf{ioV}^\infty$ *contain the theory* ioVNC^1.

Proof of theorem 26. We only need to derive $\Sigma_0^B(\mathsf{MBBFE})$-CA in $t(n)\text{-}\mathsf{ioV}^\infty$. Our goal is to show that every instance of $\Sigma_0^B(\mathsf{MBBFE})$-CA can be proven in $t(n)\text{-}\mathsf{ioV}^\infty$. Let φ be an arbitrary Σ_0^B formula, s a term, and A the free variable in φ from an instance of $\Sigma_0^B(\mathsf{MBBFE})$-CA. Recall that φ represents the graph of an AC^0 function. Abusing the notation, we use $\varphi(A)$ to denote this function. We can prove the existence of the $Z = \varphi(A)$ using the comprehension axiom in ioV^0. Note that s is a number term bounded by a polynomial in $|A|$. Let $n = |A|$. We show that there is a $\Sigma_\infty^{B \leq t(n)}$ formula $\psi(x, A)$ which provably gives the bit graph of the computation Y of circuit Z in that instance of $\Sigma_0^B(\mathsf{MBBFE})$-CA for φ and s. In other words, $\psi(x, A)$ holds iff the value computed for gate x of the circuit Z is one. It is straightforward to show that Z is unique if it exists. The existence of Z follows from applying $\Sigma_\infty^{B \leq t(n)}$-CA for ψ which describes the computation of the circuit Z.

Let Z be a balanced Boolean formula of size s and x a gate in Z. Formula ψ is similar to Buss's ALogTime algorithms for the Boolean Formula Evaluation problem [Bus87; Bus93]. We describe it as a d-round game between two players where d is a natural number we will fix later. The goal of the first player is to show that the correct value of gate x in Z is 1 while the second player's goal is to show that is not the case. We refer to them as P(rover) and C(hallenger).

We divide Z into d levels of equal height. This results in subformulas of size $O(s^{1/d})$. We look at each of these small subformulas as a possible round in the game. The game

tree has depth d and branching factor $O(s^{1/d})$. 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 numbers $w = (i_1, i_2, \ldots, i_l)$ where $0 \leq l < d$ and each number in the sequence is less than $s^{1/d}$. For example, the subformula in the root is indexed as $Z_{()}$. The subformulas below it are indexed 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 leaf.

The game starts with P giving a computation of the top subformula including the inputs to it. 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 game continues by moving to that subformula. The game ends when we reach the original input bits. At that point we can check if all claims by P are correct:

- The values for gates in each subcircuit are consistent.
- The value of the output gate of each subcircuit is equal to the value of the challenged input bit of the upper subcircuit.

If all claims by P were correct, P wins; otherwise, C wins.

The player P represents existential string quantifiers of size $O(s^{1/d})$. The player C represents universal number quantifiers of size $O(s^{1/d})$. Note that given a circuit, an input, and a computation, the fact that computation is correct is expressible as a Σ_0^B formula as in definition 66. We have d blocks of these quantifier blocks followed by a Σ_0^B formula that checks if the claims are correct:

- For each subformula in a game, the computation given for that subformula is compatible with the gates for that subformula.
- The value of the root for each subformula is equal to the value of the leaf challenged in the previous round by C.
- Gate x belongs to one of the subformulas in the game.
- The value of gate x is 1.

If we pick $d = \frac{\lg s}{\varepsilon \lg n}$, then $s^{1/d} \leq n^{\varepsilon} = t(n)$ and ψ is a $\Sigma_\infty^{B \leq t(n)}$ 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 $\Sigma_\infty^{B \leq t(n)}$ 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 achieved by case analysis: either the gate is inside a subformula, in which case the claim holds, or it is on the border between two levels, in which case it is assigned the same values in both subformula computations. This completes the proof. □

8.4 Simulating Frege by Bounded-depth Frege

In this section we combine the results from previous sections to provide an alternative proof of [FPS15] that polynomial-size Frege proofs can be simulated by subexponential-size bounded-depth Frege proofs. In other words, Frege($n^{O(1)}$) $\subseteq$ Frege($2^{O(t(n))}, O(1)$) and subexponential-size bounded-depth Frege proofs simulate polynomial-size Frege proofs. Recall that $t(n) = n^\varepsilon$.

Theorem 27. $t(n)$-ioV$^\infty \vdash$ Snd(Frege).

Proof. Combine $t(n)$-ioV$^\infty \vdash$ ioVNC^1 from theorem 26 with ioVNC$^1 \vdash$ Snd(Frege) from theorem 25. □

Corollary 6. *The proof class polynomial-size* $t(n)$-bdG$_\infty$ *proves the soundness of the* Frege *proof system.*

Proof. This follows from the fact that a $t(n)$-ioV$^\infty$ proof can be translated into a proof family in $t(n)$-bdG$_\infty$ by theorem 20. By the previous theorem $t(n)$-ioV$^\infty$ proves the soundness of Frege; therefore, $t(n)$-bdG$_\infty$ proves the propositional translation of the soundness of Frege. □

Corollary 7. *The proof class polynomial-size* $t(n)$-bdG$_\infty$ *contains the proof class polynomial-size* Frege. G *effectively simulates* Frege *by polynomial-size* $t(n)$-bdG$_\infty$ *proofs.*

Proof. This follows from corollary 6 and the weak universality of soundness from theorem 7. □

Chapter 9

Conclusion

9.1 Open Problems and Future Directions

In this section we mention a few directions that can be pursued.

Question 1. *Design theories for complexity classes inside* AC^0 *like* AC^0_2 *or polynomial size CNFs.*

The interest in these classes comes from their connections to heuristic *SAT* algorithms used in industry. Note that these classes are not closed under composition. The io-typed framework can prove a foundation to design theories for these classes.

Question 2. *Design a theory for* NP*.*

If $\mathsf{NP} \neq \mathsf{coNP}$, then NP is not closed under composition. Again the io-typed framework may provide a foundation to design a theory.

Question 3. *What makes a propositional translation reasonable?*

Note that when translating from uniform proof complexity to nonuniform proof complexity we have to translate both formulas and proofs. The usual interest in proof complexity is to study the relationship between the size of the proof and the size of tautologies. If we translate formulas in an unusual way to be large, the translation of proofs will still work. However, the relationship between the size of the proofs and the size of the tautologies will be very different. The reductions between proof search problems seem a more appealing venue than the general p-simulation definition in [Rec76].

Question 4. *Can we obtain proof complexity lower bounds when the structure of the proofs are restricted?*

The typical restriction is requiring the proof to be a tree in place of a dag. But it looks like that there can be more restrictions on the structure of the proof that may lead to new lower bounds for proof complexity classes that we do not have lower bounds for, the prime example being $\mathsf{AC}^0[p]$-Frege. Note that we have exponential lower bounds for the corresponding complexity class. The intuition here is that a proof builds a formula requiring a high complexity function to witness by composing functions through cut rules. The restriction on cut rules limits the functions we can compose but it might not be sufficient to restrict the proof enough for them to correspond to the complexity classes of interest like $\mathsf{AC}^0[p]$. A first idea might be to bound the length of "dependent cut chains" in the proof. However, the naive way of imposing such a restriction does not work by itself as several cuts can be replaced with a single cut:

$$\dfrac{\dfrac{\Gamma \Rightarrow \varphi \quad \varphi \Rightarrow \psi}{\Gamma \Rightarrow \psi}\,\textbf{Cut} \qquad \psi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\,\textbf{Cut}$$

$$\dfrac{\dfrac{}{\Rightarrow (\varphi \to \varphi) \wedge (\psi \to \psi)} \qquad \dfrac{\dfrac{\dfrac{\Gamma \Rightarrow \varphi \quad \varphi \Rightarrow \psi}{\Gamma, \varphi \to \varphi \Rightarrow \psi} \qquad \psi \Rightarrow \Delta}{\Gamma, \varphi \to \varphi, \psi \to \psi \Rightarrow \Delta}}{\Gamma, (\varphi \to \varphi) \wedge (\psi \to \psi) \Rightarrow \Delta}}{\Gamma \Rightarrow \Delta}\,\textbf{Cut}$$

Question 5. *Can we transfer diagonalization arguments to the context of proof complexity?*

Consistency statements can be used for separating theories, e.g. $\mathsf{I}\Delta_0 + \mathsf{Superexp}$ can prove the consistency of $\mathsf{I}\Delta_0$ [HP93, Theorem 5.20] while $\mathsf{I}\Delta_0 + \mathsf{Exp}$ and weaker theories cannot prove the consistency of $\mathsf{I}\Delta_0$ [WP87]. This can be considered diagonalization results in the context of proof complexity. However, this and similar results are rather weak. Can we have hierarchy results corresponding to time, space, and depth hierarchy in proof complexity? A restricted form of soundness for proofs whose lines can be evaluated in the theory can be a good candidate. Buss's BQCon [Bus86; Pud90] does not bound the terms and therefore the formulas in such a proof cannot be evaluated uniformly since the running time would not be bounded. However, for proofs with a fixed polynomial upper bound on the size of their terms, we can evaluate formulas in the proofs. A more restricted form of consistency statements related to subsystems of G was studied in [KT92]. See [Kra95, §10.5] for an exposition.

This is also interesting since it implies that the main obstacle that prevents a system

from proving its own consistency in Gödel's incompleteness theorem is the undefinability of a global truth for the formulas in the proofs. In fact, even in the case of PA if we restrict the quantifier complexity of the formulas in the proofs to Σ_k we can define truth and prove their soundness inside PA. In this sense, what is behind Gödel's incompleteness theorems is Tarski's undefinability of (global) truth theorem.

It seems possible that if we pick complexity classes corresponding to the time hierarchy theorem and design theories for them, a restricted form of soundness can separate them and give a fine hierarchy of bounded arithmetic theories. A finer diagonalization proof complexity result can lead to the transfer of other more advanced tools from computational complexity theory to proof complexity.

Another direction for obtaining a proof complexity version of diagonalization is to use finitistic consistency statements [Pud85; Pud86]. See [Pud96] and [Kra95, §14.1] for a survey of results and [Kra04a] for further developments in this direction.

Question 6. *Is there a counterpart to Allender-Koucký for* TC^0-Frege *and* NC^1-Frege*? Can we prove that to separate polynomial-size* TC^0-Frege *from polynomial-size* NC^1-Frege *it suffices to prove an* $n^{1+\varepsilon}$ *(or a similar fixed polynomial) lower bound on the size of* TC^0-Frege *proofs of the soundness of* Frege*? Can we provide a self-reducibility argument for the soundness of* Frege*? Can we prove such a result assuming plausible complexity conjectures like* $\mathsf{TC}^0 \neq \mathsf{NC}^1$*?*

The key idea behind the result in [AK10] is exploiting the self-reducibility of BFE: There is a linear-size TC^0 circuit with $\mathsf{BFE}_{\sqrt{n}}$ oracles for BFE_n. Repeating this d times we obtain a linear-size TC^0 circuit with $\mathsf{BFE}_{\sqrt[2^d]{n}}$ oracles and the result follows from choosing a suitable constant d.

Soundness statements for proof classes can be considered as counterparts to circuit evaluation problems. The goal is to prove a self-reducibility argument for the soundness of Frege similar to the self-reducibility of BFE. The first attempt to show that soundness tautologies are self-reducible fails because it involves cuts over the truth of NC^1 formulas which are not available in TC^0-Frege. We can show that a derivational form of soundness is strongly self-reducible with respect to the number of lines (not total size). This cannot reduce the size of the proof below the size of a line and therefore we need a self-reduction for the rules. In particular, we need a linear-size strong self-reduction for the NC^1-cut rule in TC^0-Frege.

Note that a single step of cut elimination process is not a strong self-reduction. Moreover, we may need to duplicate the side formulas (because of contractions in the proof). We see this as the main blocking obstacle to a proof complexity version of [AK10].

Another class of proof systems is based on algebra and geometry objects, e.g. Nullstellensatz, CP, LS, SA, SoS.

Question 7. *Design uniform systems for algebraic proof systems and establish a similar correspondence between the uniform and nonuniform versions.*

In our translation we assumed that the size of the translation of the formulas is at least linear in the size of their free variable by padding the formulas with disjunctions with $\bot$. This was required to make the general theorem about the relation of the size of translated proofs and the size of the translated formulas hold. It seems that we need some conditions on the translation of the formulas to make sure they are not so succinct that the translation of axioms about them are useless, e.g. we do not end up with a formula φ containing a function symbol f whose proof requires an axiom about f while the translation of the axiom is significantly larger than the translation of the formula. However, we have not been able to come up with such an example.

Question 8. *Is there a bounded arithmetic theory T extending V^0 with a propositional translation such that there is a provable formula in T whose translation does not have polynomial-size bounded-depth H proofs from the translation of the additional axioms of T?*

As we discussed in section 1.3, the correspondence between uniform and nonuniform systems is not perfect. Several uniform theories may correspond to the same nonuniform class. One way of dealing with this issue might be to find suitable categories on each side and show that the correspondence is an adjunction between the two categories. For example, it is possible to define a category of bounded arithmetic theories and a category of proof complexity classes such that the propositional translation is a functor from the former to the latter. Is it possible to do so in a way that propositional translation has a left adjoint? In other words, is there a consistent mapping of proof complexity classes to bounded arithmetic theories?

Question 9. *Explore various proof complexity constructions from a category-theoretic perspective.*

Proving the soundness of proofs with larger size is easier. This seems unsatisfactory. Similar to computational complexity, it would be interesting to see if we can prove the soundness of proofs when the proof is given as an oracle (i.e. function symbol). We may still evaluate the formulas in the proof, however, our induction might not be strong enough to go over a large proof. Can we restrict these large proofs in such way that their soundness in oracle form becomes provable? Is there a connection to *implicit proofs* of [Kra04b]?

Question 10. *Define a nice subclass of subexponential-size bounded-depth* Frege *proofs containing polynomial-size* n^{ϵ}-bdG_{∞} *and show that their soundness is still provable even when the proofs are given as oracles.*

Finally, I would like to state an informal question in proof complexity with Kreiselian motifs which I find most fascinating:

Question 11. *What do we gain from having a proof of a tautology in addition to knowing that it is true? What do we gain from having a proof which uses a restricted class of concepts aside from its mathematical and philosophical appeal?*

The provability of the correctness of an algorithm in a proof system is an implicit limitation on the power of the algorithm. E.g. a SAT algorithm which is provably sound in a proof system cannot solve tautologies which are hard for the proof system [Kra12]. Is there a positive side to the provability of the soundness of an algorithm in a weak proof system? E.g. can we use such proofs for automatic failure detection and recovery?

Bibliography

[Aeh10] Klaus Aehlig. "Parallel time and proof complexity". PhD thesis. University of Munich, 2010.

[Ajt83] Miklós Ajtai. "Σ_1^1-formulae on finite structures". In: *Annals of pure and applied logic* (1983), pp. 1–48.
doi: 10.1016/0168-0072(83)90038-6.

[Ajt88] Miklós Ajtai. "The complexity of the pigeonhole principle". In: *Foundations of computer science* (1988), pp. 346–355.
doi: 10.1007/BF01302964.

[All99] Eric Allender. "The permanent requires large uniform threshold circuits". In: *Chicago journal of theoretical computer science* (1999), pp. 127–135.
doi: 10.1007/3-540-61332-3_145.

[AK10] Eric Allender and Michal Koucký. "Amplifying lower bounds by means of self-reducibility". In: *Journal of the ACM* (2010), pp. 141–1436.
doi: 10.1145/1706591.1706594.

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
doi: 10.1017/cbo9780511804090.007.

[BS14a] Boaz Barak and David Steurer. "Sum-of-squares proofs and the quest toward optimal algorithms". In: *Arxiv preprint arxiv:1404.5236* (2014), pp. 1–27.

[BCE+95] Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. "The relative complexity of NP search problems". In: *Symposium on theory of computing* (1995), pp. 303–314.
doi: 10.1145/225058.225147.

[BIK+92] Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, Pavel Pudlák, and Alan Woods. "Exponential lower bounds for the pigeonhole principle". In: *Symposium on theory of computing* (1992), pp. 200–220. doi: 10.1145/129712.129733.

[BKS04] Paul Beame, Henry Kautz, and Ashish Sabharwal. "Towards understanding and harnessing the potential of clause learning". In: *Journal of artificial intelligence research* (2004), pp. 319–351. doi: 10.1613/jair.1410.

[BS14b] Paul Beame and Ashish Sabharwal. "Non-restarting SAT solvers with simple preprocessing can efficiently simulate resolution". In: *AAAI conference on artificial intelligence* (2014), pp. 2608–2615.

[BC92] Stephen Bellantoni and Stephen A. Cook. "A new recursion-theoretic characterization of the polytime functions". In: *Computational complexity* (1992), pp. 97–110. doi: 10.1007/bf01201998.

[BDG+04] Maria Luisa Bonet, Carlos Domingo, Ricard Gavalda, Alexis Maciel, and Toniann Pitassi. "Non-automatizability of bounded-depth Frege proofs". In: *Computational complexity* (2004), pp. 47–68. doi: 10.1007/s00037-004-0183-5.

[BPR00] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. "On interpolation and automatization for Frege systems". In: *SIAM journal on computing* (2000), pp. 1939–1967. doi: 10.1137/s0097539798353230.

[BCP83] Allan Borodin, Stephen A. Cook, and Nicholas Pippenger. "Parallel computation for well-endowed rings and space-bounded probabilistic machines". In: *Information and control* (1983), pp. 113–136. doi: 10.1016/s0019-9958(83)80060-6.

[Bus86] Samuel R. Buss. *Bounded arithmetic*. Bibliopolis, 1986.

[Bus87] Samuel R. Buss. "The Boolean formula value problem is in ALogTime". In: *Symposium on theory of computing* (1987), pp. 123–131. doi: 10.1145/28395.28409.

[Bus93] Samuel R. Buss. "Algorithms for Boolean formula evaluation and for tree-contraction". In: *Proof theory, complexity, and arithmetic*. Oxford University Press, 1993, pp. 95–115.

[CF10] Yijia Chen and Jörg Flum. "On p-optimal proof systems and logics for PTime". In: *International colloquium on automata, languages and programming* (2010), pp. 321–332.
doi: 10.1007/978-3-642-14162-1_27.

[CT12] Eden Chlamtac and Madhur Tulsiani. "Convex relaxations and integrality gaps". In: *Handbook on semidefinite, conic and polynomial optimization*. Springer, 2012, pp. 139–169.
doi: 10.1007/978-1-4614-0769-0_6.

[CK02] Peter Clote and Evangelos Kranakis. *Boolean functions and computation models*. Springer, 2002.
doi: 10.1007/978-3-662-04943-3_1.

[CT86] Peter Clote and Gaisi Takeuti. "Exponential time and bounded arithmetic". In: *Structure in complexity theory* (1986), pp. 125–143.
doi: 10.1007/3-540-16486-3_94.

[Coo71] Stephen A. Cook. "The complexity of theorem-proving procedures". In: *Symposium on theory of computing* (1971), pp. 151–158.
doi: 10.1145/800157.805047.

[Coo75] Stephen A. Cook. "Feasibly constructive proofs and the propositional calculus". In: *Symposium on theory of computing* (1975), pp. 83–97.
doi: 10.1145/800116.803756.

[Coo12] Stephen A. Cook. "Relativized propositional calculus". In: *Arxiv preprint arxiv:1203.2168* (2012), pp. 1–8.

[CM05] Stephen A. Cook and Tsuyoshi Morioka. "Quantified propositional calculus and a second-order theory for NC1". In: *Archive for mathematical logic* (2005), pp. 711–749.
doi: 10.1007/s00153-005-0282-2.

[CN10] Stephen A. Cook and Phoung Nguyen. *Logical foundations of proof complexity*. Cambridge University Press, 2010.
doi: 10.1017/cbo9780511676277.

[CR74] Stephen A. Cook and Robert Reckhow. "On the lengths of proofs in the propositional calculus". In: *Symposium on theory of computing* (1974), pp. 135–148.
doi: 10.1145/800119.803893.

[CR79] Stephen A. Cook and Robert Reckhow. "The relative efficiency of propositional proof systems". In: *The journal of symbolic logic* (1979), pp. 36–50. doi: 10.2307/2273702.

[CS99] Stephen A. Cook and Michael Soltys. "Boolean programs and quantified propositional proof systems". In: *Bulletin of symbolic logic* (1999), pp. 119–129. doi: 10.1.1.150.2814.

[CU93] Stephen A. Cook and Alasdair Urquhart. "Functional interpretations of feasibly constructive arithmetic". In: *Symposium on theory of computing* (1993), pp. 107–112. doi: 10.1145/73007.73017.

[Dal12] Ugo Dal Lago. "A short introduction to implicit computational complexity". In: *Lectures on logic and computation*. Springer, 2012, pp. 89–109. doi: 10.1007/978-3-642-31485-8_3.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. "A machine program for theorem-proving". In: *Communications of the ACM* (1962), pp. 394–397. doi: 10.1145/368273.368557.

[DP60] Martin Davis and Hilary Putnam. "A computing procedure for quantification theory". In: *Journal of the ACM* (1960), pp. 201–215. doi: 10.1145/321033.321034.

[Edm65] Jack Edmonds. "Paths, trees, and flowers". In: *Canadian journal of mathematics* (1965), pp. 449–467. doi: 10.4153/cjm-1965-045-4.

[Emd90] Peter van Emde Boas. "Machine models and simulations". In: *Handbook of theoretical computer science, vol. A*. Elsevier, 1990, pp. 1–66. doi: 10.1016/b978-0-444-88071-0.50006-0.

[FPS15] Yuval Filmus, Toniann Pitassi, and Rahul Santhanam. "Exponential lower bounds for AC0-Frege imply superpolynomial Frege lower bounds". In: *Transactions on computation theory* (2015), pp. 618–629. doi: 10.1145/2656209.

[FH03] Lance Fortnow and Steve Homer. "A short history of computational complexity". In: *Conference on computational complexity* (2003), pp. 95–133. doi: 10.1109/ccc.2002.1004340.

[FLM+05] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas. "Time-space lower bounds for satisfiability". In: *Journal of the ACM* (2005), pp. 835–865.
doi: 10.1145/1101821.1101822.

[FSS84] Merrick Furst, James B Saxe, and Michael Sipser. "Parity, circuits, and the polynomial-time hierarchy". In: *Mathematical systems theory* (1984), pp. 13–27.
doi: 10.1007/bf01744431.

[GC13] Kaveh Ghasemloo and Stephen A. Cook. "Theories for subexponential-size bounded-depth Frege proofs." In: *Computer science logic* (2013), pp. 296–315.
doi: 10.4230/LIPIcs.CSL.2013.296.

[Gol12] Oded Goldreich. *Every NL set has constant-depth Boolean circuits of sub-exponential size*. 2012.

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. "Boosting combinatorial search through randomization". In: *AAAI innovative applications of artificial intelligence* (1998), pp. 431–437.

[GHP02] Dima Grigoriev, Edward A. Hirsch, and Dmitrii V. Pasechnik. "Complexity of semi-algebraic proofs". In: *Symposium on theoretical aspects of computer science* (2002), pp. 419–430.
doi: 10.1007/3-540-45841-7_34.

[HP93] Petr Hájek and Pavel Pudlák. *Metamathematics of first-order arithmetic*. Springer, 1993, pp. 295–297.
doi: 10.1007/978-3-662-22156-3.

[Hak85] Armin Haken. "The intractability of resolution". In: *Theoretical computer science* (1985), pp. 297–308.
doi: 10.1016/0304-3975(85)90144-6.

[HS65] Juris Hartmanis and Richard E. Stearns. "On the computational complexity of algorithms". In: *Transactions of the AMS* (1965), pp. 285–306.
doi: 10.2307/1994208.

[Hås87] Johan Håstad. "Computational limitations of small-depth circuits". PhD thesis. Massachusetts Institute of Technology, 1987.

[Ign95] Aleksandar Ignjatović. "Delineating classes of computational complexity via second order theories with weak set existence principles I". In: *The journal of symbolic logic* (1995), pp. 103–121.
doi: 10.2307/2275511.

[Imm99] Neil Immerman. *Descriptive complexity*. Springer, 1999.
doi: 10.1007/978-1-4612-0539-5.

[Jeř05] Emil Jeřábek. "Weak pigeonhole principle, and randomized computation". PhD thesis. Charles University in Prague, 2005.

[Juk12] Stasys Jukna. *Boolean function complexity: advances and frontiers*. Springer, 2012.
doi: 10.1007/978-3-642-24508-4.

[KC96] Bruce M. Kapron and Stephen A. Cook. "A new characterization of type-2 feasibility". In: *Foundations of computer science* (1996), pp. 117–132.
doi: 10.1109/sfcs.1991.185389.

[KL82] Richard M. Karp and Richard J. Lipton. "Turing machines that take advice". In: *Logic and algorithmic* (1982), pp. 191–209.
doi: 10.5169/seals-52237.

[KSM11] Hadi Katebi, Karem A. Sakallah, and Joao P. Marques-Silva. "Empirical study of the anatomy of modern sat solvers". In: *Theory and applications of satisfiability testing* (2011), pp. 343–356.
doi: 10.1007/978-3-642-21581-0_27.

[Kol05] Antonina Kolokolova. "Systems of bounded arithmetic from descriptive complexity". PhD thesis. University of Toronto, 2005.

[Kra95] Jan Krajíček. *Bounded arithmetic, propositional logic, and complexity theory*. Cambridge University Press, 1995.
doi: 10.1017/cbo9780511529948.

[Kra04a] Jan Krajíček. "Diagonalization in proof complexity". In: *Fundamenta mathematicae* (2004), pp. 181–192.
doi: 10.4064/fm182-2-7.

[Kra04b] Jan Krajíček. "Implicit proofs". In: *The journal of symbolic logic* (2004), pp. 387–397.
doi: 10.2178/jsl/1082418532.

[Kra10] Jan Krajíček. *Forcing with random variables and proof complexity*. Cambridge University Press, 2010.
doi: 10.1017/CBO9781139107211.

[Kra12] Jan Krajíček. "A note on SAT algorithms and proof complexity". In: *Information processing letters* (2012), pp. 490–493.
doi: 10.1016/j.ipl.2012.03.009.

[KP89] Jan Krajíček and Pavel Pudlák. "Propositional proof systems, the consistency of first order theories and the complexity of computations". In: *The journal of symbolic logic* (1989), pp. 1063–1079.
doi: 10.2307/2274765.

[KP90] Jan Krajíček and Pavel Pudlák. "Quantified propositional calculi and fragments of bounded arithmetic". In: *Mathematical logic quarterly* (1990), pp. 29–46.
doi: 10.1002/malq.19900360106.

[KPW95] Jan Krajíček, Pavel Pudlák, and Alan Woods. "An exponential lower bound to the size of bounded-depth Frege proofs of the pigeonhole principle". In: *Random structures & algorithms* (1995), pp. 15–39.
doi: 10.1002/rsa.3240070103.

[KT92] Jan Krajíček and Gaisi Takeuti. "On induction-free provability". In: *Annals of mathematics and artificial intelligence* (1992), pp. 107–125.
doi: 10.1007/bf01531024.

[Lad75] Richard E. Ladner. "The Circuit Value Problem is LogSpace-complete for P". In: *ACM SIGACT news* (1975), pp. 18–20.
doi: 10.1145/990518.990519.

[Lê14] Dai Tri Man Lê. "Bounded arithmetic and formalizing probabilistic proofs". PhD thesis. University of Toronto, 2014.

[Lei91] Daniel Leivant. "A foundational delineation of computational feasibility". In: *Logic in computer science* (1991), pp. 2–11.
doi: 10.1109/LICS.1991.151625.

[Lev73] Leonid A. Levin. "Universal sequential search problems". In: *Problems of information transmission* (1973), pp. 115–116.

[MLM09] Joao Marques-Silva, Ines Lynce, and Sharad Malik. "Conflict-driven clause learning SAT solvers". In: *Handbook of satisfiability*. IOS Press, 2009, pp. 131–153.
doi: 10.3233/978-1-58603-929-5-131.

[Mor05] Tsuyoshi Morioka. "Logical approaches to the complexity of search problems: proof complexity, quantified propositional calculus, and bounded arithmetic". PhD thesis. University of Toronto, 2005.

[Mül13] Sebastian Müller. "Polylogarithmic cuts in models of V0". In: *Logical methods in computer science* (2013), p. 2013.
doi: 10.2168/LMCS-9(1:16)2013.

[Nep70] V.A. Nepomnjascij. "Rudimentary predicates and Turing calculations". In: *Cybernetics* (1970), pp. 212–218.
doi: 10.1007/bf01069074.

[Ngu08] Phuong Nguyen. "Bounded reverse mathematics". PhD thesis. University of Toronto, 2008.

[Par71] Rohit Parikh. "Existence and feasibility in arithmetic". In: *The journal of symbolic logic* (1971), pp. 494–508.
doi: 10.2307/2269958.

[Per09] Steven Perron. "Power of non-uniformity in proof complexity". PhD thesis. University of Toronto, 2009.

[Pic14] Ján Pich. "Complexity theory in feasible mathematics". PhD thesis. Charles University in Prague, 2014.

[Pic15] Ján Pich. "Circuit lower bounds in bounded arithmetics". In: *Annals of pure and applied logic* (2015), pp. 29–45.
doi: 10.1016/j.apal.2014.08.004.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. "On the power of clause-learning SAT solvers as resolution engines". In: *Artificial intelligence* (2011), pp. 512–525.
doi: 10.1016/j.artint.2010.10.002.

[Pip79] Nicholas Pippenger. "On simultaneous resource bounds". In: *Foundations of computer science* (1979), pp. 307–311.
doi: 10.1109/sfcs.1979.29.

[Pit92] Toniann Pitassi. "The complexity of weak formal systems". PhD thesis. University of Toronto, 1992.

[PBI93] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. "Exponential lower bounds for the pigeonhole principle". In: *Computational complexity* (1993), pp. 97–140.
doi: 10.1007/bf01200117.

[Pit07] François Pitt. "A quantifier-free string theory for ALogTime reasoning". PhD thesis. University of Toronto, 2007.

[Pud85] Pavel Pudlák. "Improved bounds to the lengths of proofs of finitistic consistency statements". In: *Contemporary mathematics* (1985), pp. 309–331.
doi: 10.1090/conm/065/891256.

[Pud86] Pavel Pudlák. "On the length of proofs of finitistic consistency statements in first order theories". In: *Studies in logic and the foundations of mathematics* (1986), pp. 165–196.
doi: 10.1016/s0049-237x(08)70462-2.

[Pud90] Pavel Pudlák. "A note on bounded arithmetic". In: *Fundamenta mathematicae* (1990), pp. 85–89.

[Pud96] Pavel Pudlák. "On the lengths of proofs of consistency". In: *Collegium logicum*. Springer, 1996, pp. 65–86.
doi: 10.1007/978-3-7091-9461-4_5.

[Rab60] Michael O. Rabin. "Degree of difficulty of computing a function and a partial ordering of recursive sets". PhD thesis. Hebrew University, Jerusalem, Israel, 1960.

[Raz87] Alexander A. Razborov. "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition". In: *Mathematical notes* (1987), pp. 333–338.
doi: 10.1007/BF01137685.

[Raz95a] Alexander A. Razborov. "Bounded arithmetic and lower bounds in Boolean complexity". In: *Feasible mathematics II*. Springer, 1995, pp. 344–386.
doi: 10.1007/978-1-4612-2566-9_12.

[Raz95b] Alexander A. Razborov. "Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic". In: *Izvestiya: mathematics* (1995), pp. 205–227.
doi: 10.1070/im1995v059n01abeh000009.

[RR94] Alexander A. Razborov and Steven Rudich. "Natural proofs". In: *Symposium on theory of computing* (1994), pp. 204–213.
doi: 10.1006/jcss.1997.1494.

[Rec76] Robert A. Reckhow. "On the lengths of proofs in the propositional calculus". PhD thesis. University of Toronto, 1976.

[Rob65] John A. Robinson. "A machine-oriented logic based on the resolution principle". In: *Journal of the ACM* (1965), pp. 23–41.
doi: 10.1145/321250.321253.

[Ruz79] Walter L. Ruzzo. "On uniform circuit complexity". In: *Foundations of computer science* (1979), pp. 312–318.
doi: 10.1109/sfcs.1979.31.

[Sch76] Claus-Peter Schnorr. "The network complexity and the Turing machine complexity of finite functions". In: *Acta informatica* (1976), pp. 95–107.
doi: 10.1007/bf00265223.

[Sim09] Stephen George Simpson. *Subsystems of second order arithmetic*. Cambridge University Press, 2009.
doi: 10.1017/cbo9780511581007.

[Ske06] Alan Ramsay Skelley. "Theories and proof systems for pspace and the exp-time hierarchy". PhD thesis. University of Toronto, 2006.

[Smo87] Roman Smolensky. "Algebraic methods in the theory of lower bounds for Boolean circuit complexity". In: *Symposium on theory of computing* (1987), pp. 77–82.
doi: 10.1145/28395.28404.

[Sol01] Michael Soltys. "Proof complexity of linear algebra". PhD thesis. University of Toronto, 2001.

[Spi71] Philip M. Spira. "On time-hardware complexity tradeoffs for Boolean functions". In: *Hawaii symposium on system sciences* (1971), pp. 525–527.

[Tse68] Grigori S. Tseitin. "On the complexity of derivation in propositional calculus". In: *Automation of reasoning 2*. 1968, pp. 115–125.
doi: 10.1007/978-3-642-81955-1_28.

[UF96] Alasdair Urquhart and Xudong Fu. "Simplified lower bounds for propositional proofs". In: *Notre Dame journal of formal logic* (1996), pp. 523–544.
doi: 10.1305/ndjfl/1040046140.

[Vol99] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer, 1999.
doi: 10.1007/978-3-662-03927-4.

[WP87] Alex J Wilkie and Jeff B Paris. "On the scheme of induction for bounded arithmetic formulas". In: *Annals of pure and applied logic* (1987), pp. 261–302.
doi: 10.1016/0168-0072(87)90066-2.

[Zam96] Domenico Zambella. "Notes on polynomially bounded arithmetic". In: *The journal of symbolic logic* (1996), pp. 942–966.
doi: 10.2307/2275794.