Informatics

Sequent Calculi for QBFs

Their Relation to Bounded Arithmetic, and the Complexity of the Witnessing Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computational Intelligence

by

Dana Jomar
Registration Number 1426666

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ-Prof. Dr. Uwe Egly

Vienna, 31st May, 2021

Dana Jomar

Uwe Egly

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Dana Jomar

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Mai 2021

Dana Jomar

Acknowledgements

I wish to express my sincere gratitude to my thesis advisor Prof. Dr. Uwe Egly. He was the compass that always steered me in the right direction. For his support, guidance, and above all his patience, I would like to say thank you.

It is also essential to express my gratitude to the people who encouraged me, and gave the emotional support I needed to continue when I thought I will never finish. To you all – you know who you are – thank you.

Abstract

G_i and G_i^* are Gentzen-like sequent calculi for QBFs, in which applications of the cut rule are restricted to $\Sigma_i^q \cup \Pi_i^q$-formulas. The target formula of a quantifier rule in these calculi is restricted to quantifier-free formulas. By G_i^*-proofs we denote treelike G_i-proofs. The systems G_i and G_i^* are closely related to the theories S_2^i and T_2^i of bounded arithmetic [Bus86]. In fact a proof of a bounded arithmetic formula can be translated to a proof of a QBF. The systems G_0 and G_0^*, in which cut formulas are restricted to quantifier-free formulas (i.e., propositional), are similar to first-order theories T axiomatized by purely universal formulas. This allows us to explore results proven for T like Herbrands theorem and the midsequent theorem on G_0 and G_0^*.

For a proof system H we define the Σ_k^q-Witnessing problem to be as follows. Given a Σ_k^q-formula, an H-proof of it, and a truth assignment to the free variables, find a witness for the outermost existentially quantified variables. Morioka proved in [Mor05], also presented with Cook in [CM05], that the Σ_1^q-Witnessing problem for G_0 and G_0^* is $\mathbf{NC^1}$-complete. We bring in this thesis all of the elements which were used in the proof together, and present the end result to the reader.

Contents

CHAPTER 1

Introduction

Quantified Boolean formulas (QBFs) are an extension of propositional logic which allows quantification over propositional variables. The decision problem of QBFs is **PSPACE**-complete and is considered a generalization of the **NP**-complete problem SAT, the satisfiability problem of propositional formulas. Many problems from application domains such as model checking or formal verification are known to be **PSPACE**-complete, hence QBFs provide us with a powerful tool to encode them.

QBFs in prenex normal form define a hierarchy of formulas, which corresponds to the polynomial hierarchy (**PH**). This implies that any problem that falls into a problem class in **PH** can also be represented by a QBF. The hierarchy of QBFs in prenex normal form is based on the number of existential and universal quantifier alternations in a formula. It helps to think of QBFs as a game between two players $\exists$ and $\forall$. In the current outermost (leftmost) quantifier block, the corresponding player has the right to choose an assignment for all the variables in this block. The chosen assignment is applied to the formula, the formula simplifies under the assignment and the outermost quantifier block disappears, resulting in the possibility for a move of the other player. The $\exists$ player will guess an assignment to the existentially quantified variables trying to satisfy the formula, and the $\forall$ player will guess an assignment to the universally quantified variables and tries to let the QBF evaluate to false. The $\forall$ player has a winning strategy if and only if the original formula is false.

Krajíček and Pudlák introduced in [KP90] the sequent calculi KPG for QBFs, and defined a hierarchy of fragments KPG$_i$, KPG$_i^*$, such that a KPG$_i$-proof is restricted to $\Sigma_i^q \cup \Pi_i^q$-formulas, and KPG$_i^*$ is KPG$_i$ but restricted to treelike proofs. Those systems are closely related to the theories S_2^i and T_2^i of bounded arithmetic [Bus86]. By modifying the definition of those systems, Cook and Morioka in [Mor05, CM05] introduced new systems which have a better and more natural correspondence to bounded arithmetic. They introduced two restrictions. In their system G_i any QBF can be handled; thus, in contrast to the corresponding system of Krajíček and Pudlák, the new system is complete.

Furthermore, Cook and Morioka restricted the application of the cut rule to $\Sigma_i^q \cup \Pi_i^q$-formulas, and restricted the target formula in the quantifier rules to be quantifier-free. With the new definition they introduced two important new complete proof systems for proving QBFs, G_0 and G_0^*, in which the cut rule is restricted to quantifier-free (i.e., propositional) formulas. We note that KPG was originally in [KP90] denoted by G, but we use KPG to avoid confusion with the later modified calculi.

We present in this thesis the systems KPG, KPG$_i$, KPG$_i^*$, G, G_i, and G_i^* and show that corresponding systems are polynomially equivalent for proving $\Sigma_i^q \cup \Pi_i^q$-formulas. We also take a look at the relationship between QBFs and bounded arithmetic and how the systems G_i and G_i^* relate to the theories S_2^i and T_2^i.

We focus on the systems G_0 and G_0^* and we present a polynomial-time version of the midsequent theorem for G_0^* and a restricted version of the theorem for G_0, and we finally define the witnessing problem and present an $\mathbf{NC^1}$ algorithm for the Σ_1^q-Witnessing problem for G_0.

We aim with this thesis to bring the work of Cook and Morioka in [CM05] and [Mor05] closer to graduate students. As we follow their footsteps, we explore and discuss the work of Krajíček and Pudlák in QBFs and bounded arithmetic presented in [Kra95] and [KP90], till we eventually have all the results that Cook and Morioka used to reach the $\mathbf{NC^1}$-completeness results for the witnessing problem. Please note that this thesis is the first step towards the full understanding of those two references as they go beyond the border of first-order theories and prove some interesting results for second-order theories which are out of the scope of this work.

1.1 Organization

This thesis is organized as follows.

Chapter 2 presents the basics of complexity theory, circuit complexity classes and local search complexity classes, which will be used for the analysis of the witnessing problems.

Chapter 3 introduces QBFs, quantified proof systems, the definitions of the calculi KPG$_i$ (for $i \geq 0$) from [KP90] and the corresponding calculi G_i from [CM05] both in tree and dag[1] form. A detailed proof of the polynomial equivalence of corresponding KPG and G systems (for proving formulas in $\Sigma_i^q \cup \Pi_i^q$) is presented

Chapter 4 is devoted to the basics of bounded arithmetic. In this chapter we rely on [Kra95] and [KP90] to define a sequent calculus LKB for proving bounded formulas, present a translation of bounded formulas to QBFs, and prove that if we have an

[1]directed acyclic graph

(LKB + Σ_i^b-IND)-proof of a bounded formula, then we can construct a G_i-proof of its QBF translation.

Chapter 5 focuses on the restricted systems G_0 and G_0^*, where only propositional formulas can be used as cut formulas. We define the notions of π-prototypes and the Herbrand π-disjunction for a G_0-proof π, and we prove a polynomial-time version of Gentzen's midsequent theorem for G_0^*.

Chapter 6 explains the proof that the Σ_1^q-Witnessing problem is $\mathbf{FNC^1}$-complete under many-one $\mathbf{AC^0}$-reductions for both G_0 and G_0^*.

CHAPTER 2

Complexity Theory

In this chapter we introduce some complexity concepts and results, which are used in later chapters.

We start with some important complexity classes and the polynomial hierarchy.

2.1 Basic Complexity Theory

Definition 2.1.1
A *Boolean function* with n inputs and m outputs is a function

$$f : \{0,1\}^n \longmapsto \{0,1\}^m. \qquad \triangleleft$$

Boolean functions define *decision problems*, which are problems or questions with a yes/no answers. We often use 1 for yes and 0 for no.

Definition 2.1.2
A *Decision Problem* (DP) P is the Boolean function $P : \{0,1\}^* \longmapsto \{0,1\}$.

Any DP can be defined by a set of possible instances or inputs I, and a subset $I_{yes} \subseteq I$ which specifies the YES-instances of P. $\triangleleft$

Consider as an example the satisfiability problem (SAT) for propositional logic. Recall that an assignment is a mapping $I : \text{Var}(\varphi) \longmapsto \{0,1\}$, such that $\text{Var}(\varphi)$ is the set of propositional variables in φ.

Problem SAT

Input:	A Boolean formula φ.
Question:	Does there exist a satisfying assignment for φ?

Note that we restricted the function in the definition of a DP to Boolean functions (functions whose inputs are finite strings of bits, i.e., elements of $\{0,1\}^*$ and their outputs are from $\{0,1\}$). This condition is in fact not very restricting since general objects like integers, pairs, graphs, etc. can be encoded as strings of bits.

An example of such an encoding would be representing a graph G with its adjacency matrix, i.e., a graph g with n vertices is represented by an $n \times n$ matrix A, in which the element $A_{i,j}$ is 1 if and only if there is an edge between vertices i and j.

It is in fact with this in mind that we consider a language to represent an encoding of all YES-instances of a decision problem.

Definition 2.1.3
For a problem X we define the corresponding language Z_X as follows.

$$Z_X = \{e_x : e_x \text{ is the encoding of } x \ s.t. \ x \text{ is a YES-instance of } X\}. \qquad \triangleleft$$

We say that a machine *decides* or *computes* a language $L \subseteq \{0,1\}^*$ if it computes the function

$$f_L : \{0,1\}^* \longmapsto \{0,1\} \text{ where } f_L(x) = 1 \Leftrightarrow x \in L.$$

Determining if an instance of a problem is a YES-instance is equivalent to determining whether its encoding is in the corresponding language, which is why we use throughout this work language and problem interchangeably.

In 1950 Alan Turing described a simple mathematical model that is sufficient to model any computational process. In [AB09] the concept of Turing machines is described very elegantly based on the following quote from Alan Turing.

"The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer. The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail. We may suppose that these rules are supplied in a book, which is altered whenever he is put on to a new job. He has also an unlimited supply of paper on which he does his calculations" Alan Turing 1950

The idea is that a Turing machine or an algorithm for computing a function

$$f : \{0,1\}^* \mapsto \{0,1\}$$

is a set of fixed mechanical rules without a constraint on the number of times to apply each rule. Each rule can be described with a list of elementary reading and writing steps, and the running time of this machine is the number of these elementary operations. This is usually a function that depends on the length of the input string.
When this function is polynomial then we say that we have a polynomial Turing machine.

As mentioned earlier general objects can be encoded as strings of bits and that in fact includes Turing machines (a.k.a algorithms). This entails that an algorithm can be an input to another algorithm. This fact inspired the idea of a universal Turing machine that can simulate any other Turing machine.

Unless otherwise specified, we always mean universal Turing machines when talking about Turing machines.

Definition 2.1.4
Let $T : \mathbb{N} \longmapsto \mathbb{N}$ be some function.

1. A language L is in $\mathbf{DTIME(T(n))}$ if and only if it is computable with a ***deterministic*** Turing machine in $c \cdot T(n)$-time for some constant $c > 0$.
2. A language L is in $\mathbf{NTIME(T(n))}$ if and only if it is computable with a ***non-deterministic*** Turing machine in $c \cdot T(n)$-time for some constant $c > 0$. $\triangleleft$

Definition 2.1.5 (The class **P**)
The class **P** is the class of languages computable in polynomial time with a deterministic Turing machine, i.e.,

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME(n^c)}. \qquad \triangleleft$$

Definition 2.1.6 (The class **NP** – First Definition)
The class **NP** is the class of languages computable in polynomial time with a non-deterministic Turing machine, i.e.,

$$\mathbf{NP} = \bigcup_{c \geq 1} \mathbf{NTIME(n^c)}. \qquad \triangleleft$$

We note that the class **P** is the class of polynomial time solvable DPs, which roughly represents the class of feasible problems (or efficiently solvable problems), whereas the class **NP** captures the problems whose solutions might be hard to find, however they can be efficiently verified. So given an input x (the instance of the problem) we can easily verify that x is a YES instance if we are given the polynomial solution that certifies that x is a positive instance.

In the following definitions we use the notation $M(x, u)$, where M is a universal Turing machine that takes $x,\ u \in \{0,1\}^*$ as input such that if u is the encoding of a Turing machine M' then $M(x, u) = M'(x)$. A more detailed overview of Turing machines and their encodings can be found in Chapter 1 of [AB09].

A second definition of the class **NP** is as follows.

Definition 2.1.7 (The class **NP** – Second Definition)
We say that a language $L \subseteq \{0,1\}^*$ is in **NP** if there exist a polynomial $p(\cdot)$ and a deterministic polynomial time Turing machine M such that

$$\text{for every } x \in \{0,1\}^* : x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

where $|x|$ denote the string length of x.

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$ then we call u a *certificate* or a *witness* for x. ◁

Note that the first definition describes the class **NP** as the class of problems *solvable* by a nondeterministic Turing machine in polynomial time, while the second definition describes it as the class of problems *verifiable* by a deterministic Turing machine in polynomial time.

Another significant complexity class is the class **coNP**, which is defined as follows.

Definition 2.1.8 (The class **coNP**, related to the first definition of **NP**)
If $L \subseteq \{0,1\}^*$ is a language, then we denote by $\overline{L}$ the complement of L. That is, $\overline{L} = \{0,1\}^* \backslash L$.

With this we define $\mathbf{coNP} = \{L : \overline{L} \in \mathbf{NP}\}$. ◁

It is important to note that **coNP** is not the complement of **NP**. In fact they have a non-empty intersection, since every language in **P** is in $\mathbf{NP} \cap \mathbf{coNP}$.

Just like **NP**, **coNP** also has a second definition with the use of quantifiers, and while the "existence" of a witness is enough for **NP** problems, for **coNP** problems the condition must be applicable to "all", thus we can define the class as follows.

Definition 2.1.9 (The class **coNP**, related to the second definition of **NP**)
We say that a language $L \subseteq \{0,1\}^*$ is in **coNP** if there exist a polynomial $p(\cdot)$ and a deterministic polynomial time Turing machine M such that

$$\text{for every } x \in \{0,1\}^* : x \in L \Leftrightarrow \forall u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x,u) = 1. \qquad \triangleleft$$

An important concept in complexity theory is **NP**-*hardness* and **NP**-*completeness*.

Definition 2.1.10 (Reducibility)
A language L is polynomial time reducible to a language L', denoted as $L \leq_p L'$, if there is a polynomial time computable function $f : \{0,1\}^* \longmapsto \{0,1\}^*$ such that

$$\text{for every } x \in \{0,1\}^* : x \in L \text{ if and only if } f(x) \in L'. \qquad \triangleleft$$

Definition 2.1.11 (**NP**-Completeness)
We say that L' is **NP**-*hard* if $L \leq_p L'$ for every $L \in \mathbf{NP}$ (i.e., every language in **NP** can be reduced to L').

L' is **NP**-*complete* if L' is **NP**-hard and $L' \in \mathbf{NP}$. ◁

Theorem 2.1.1.

- *The operation of reducibility is transitive, i.e., if* $L \leq_p L'$ *and* $L' \leq_p L''$ *then* $L \leq_p L''$.
- *If* L *is* **NP***-hard and* $L \in \mathbf{P}$ *then* $\mathbf{P} = \mathbf{NP}$.
- *If* L *is* **NP***-complete then* $L \in \mathbf{P}$ *if and only if* $\mathbf{P} = \mathbf{NP}$.

Richard Karp proved in [Kar72] 21 **NP**-complete problems, immediately after Cook's paper [Coo71], in which he formalized the notion of polynomial-time reduction and defined the concept of **NP**-completeness.

The most significant **NP**-complete problem today is the previously defined problem SAT. This implies that if we can find a deterministic polynomial-time algorithm to decide if a Boolean formula is satisfiable then we can solve any **NP** problem deterministically and within polynomial time, which will eventually mean that $\mathbf{P} = \mathbf{NP}$.

From a practical point of view, since any **NP**-complete problem R can be efficiently reduced to SAT and since practically efficient SAT solvers exist, one can solve R by reducing it to SAT and then use a SAT solver. This procedure is widely used today, e.g., in the field of software verification.

2.1.1 The Polynomial Hierarchy

Intuitively the definition of **NP** (resp. **coNP**) implies that a problem is in **NP** (resp. **coNP**) if it is describable with "there exists" (resp. with "for all/every"), but not all problems can be represented with this simplicity.

Let us consider, for example, the INDSET problem, which is an **NP**-complete problem.

Problem INDSET	
Input:	$\langle G, k\rangle$, where G is a graph and k is an integer ≥ 1.
Question:	Does G have an independent set (a set of vertices without a common edge) of size (or rather cardinality) at least k?

The question of the INDSET problem asks about the "existence" of an independent set of size big enough, however if we modify the question to be "Is the largest independent set in G of size exactly k?", then checking the existence of the independent set is not enough. We also have to compare it with "every" other independent set, which indicates that such a problem is neither in **NP** nor in **coNP**. This motivates the following definition.

Definition 2.1.12 (The class $\mathbf{\Sigma_2^p}$)
The class $\mathbf{\Sigma_2^p}$ is defined to be the set of all languages L for which there exists a polynomial-

time Turing Machine M and a polynomial $q(\cdot)$ such that

$$\begin{aligned}&\text{for every } x \in \{0,1\}^* :\\ &\qquad x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)}\ \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1.\end{aligned}$$ ◁

Note that $\mathbf{\Sigma_2^p}$ contains **NP** and **coNP**, and similar to the idea of **coNP** we can define the class $\mathbf{\Pi_2^p}$ to contain the problems L such that

$$\begin{aligned}&\text{for every } x \in \{0,1\}^* :\\ &\qquad x \in L \Leftrightarrow \forall u \in \{0,1\}^{q(|x|)}\ \exists v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1.\end{aligned}$$

The difference between the two definitions is the change of the quantifier alternation from $\exists u \forall v$ to $\forall u \exists v$.

We generalize this definition to get the polynomial hierarchy.

Definition 2.1.13 (The Polynomial Hierarchy)
Let i be an arbitrary non-negative integer, and x an arbitrary element from $\{0,1\}^*$.

1. We say that a language L is in $\mathbf{\Sigma_i^p}$ if there exists a polynomial time Turing machine M and a polynomial $q(\cdot)$ such that

$$\begin{aligned}x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_i \in \{0,1\}^{q(|x|)}&\\ \text{s.t. } M(x,u_1,u_2,\dots,u_i) = 1,&\end{aligned}$$

   where Q_i is $\forall$ if i is even and Q_i is $\exists$ if i is odd.

2. We say that L is in $\mathbf{\Pi_i^p}$ if there exists a polynomial time Turing machine M and a polynomial $q(\cdot)$ such that

$$\begin{aligned}x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} \cdots Q_i u_i \in \{0,1\}^{q(|x|)}&\\ \text{s.t. } M(x,u_1,u_2,\dots,u_i) = 1,&\end{aligned}$$

   where Q_i is $\exists$ if i is even and Q_i is $\forall$ if i is odd.

Then the *Polynomial Hierarchy* is the set $\mathbf{PH} = \bigcup_{i \geq 0} \mathbf{\Sigma_i^p} = \bigcup_{i \geq 0} \mathbf{\Pi_i^p}$. ◁

Note that

- $\mathbf{\Sigma_0^p} = \mathbf{\Pi_0^p} = \mathbf{P}$,
- $\mathbf{\Sigma_1^p} = \mathbf{NP}$ and $\mathbf{\Pi_1^p} = \mathbf{coNP}$,
- for every i : $\mathbf{co\Sigma_i^p} = \mathbf{\Pi_i^p}$, and

- for every $i : \mathbf{\Sigma_i^p} \subseteq \mathbf{\Pi_{i+1}^p}$.

The question whether **PH** is a "real" hierarchy is related to one of the most important open question in computer science, and one of the seven Millennium problems selected for a 1 million USD prize for the first correct solution: "Does **P** equal **NP**?"

The relation of the two problems is captured by the following theorem.

Theorem 2.1.2. *If* $\mathbf{P} = \mathbf{NP}$ *then* $\mathbf{PH} = \mathbf{P}$

Proof. [AB09]

By induction on i, we prove that if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{\Sigma_i^p}, \mathbf{\Pi_i^p} \subseteq \mathbf{P}$.

Base case: $i = 1$**.**

If $\mathbf{NP} = \mathbf{P}$ then $\mathbf{\Sigma_1^p} = \mathbf{P} = \mathbf{coP} = \mathbf{coNP} = \mathbf{\Pi_1^p}$.

Induction hypothesis

For some $i \geq 1$, if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{\Sigma_i^p}, \mathbf{\Pi_i^p} \subseteq \mathbf{P}$.

Induction step

Assuming that $\mathbf{NP} = \mathbf{P}$, let $L \in \mathbf{\Sigma_{i+1}^p}$. Then by definition there is a polynomial $q(\cdot)$ such that

$$\begin{aligned} x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_{i+1} u_{i+1} \in \{0,1\}^{q(|x|)} \\ s.t.\ M(x, u_1, u_2, \ldots, u_{i+1}) = 1 \end{aligned}$$

where $Q_{i+1} \in \{\exists, \forall\}$, and Q_{i+1} is $\exists$ if $(i+1)$ is odd and Q_{i+1} is $\forall$ if $(i+1)$ is even.

Define the language L' as follows.

$$\begin{aligned} \langle x, u_1 \rangle \in L' \Leftrightarrow \forall u_2 \in \{0,1\}^{q(|x|)} \cdots Q_{i+1} u_{i+1} \in \{0,1\}^{q(|x|)} \\ s.t.\ M(x, u_1, u_2, \ldots, u_{i+1}) = 1. \end{aligned}$$

The formula representing problems in L' has $i - 1$ quantifier alternations (i.e., i quantifier blocks) starting with $\forall$ which implies that $L' \in \mathbf{\Pi_i^p}$. By the induction hypothesis we know that $\mathbf{\Pi_i^p} \subseteq \mathbf{P}$. This implies that there is a polynomial time Turing machine M' which computes L'. Then by plugging it in the definition of L we get :

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} M'(x, u_1) = 1,$$

which means that $L \in \mathbf{NP}$ which is assumed to equal $\mathbf{P}$. Therefore $L \in \mathbf{P}$.

Analogously we can prove that if $L \in \mathbf{\Pi_{i+1}^p}$ then $L \in \mathbf{P}$. However in this case we will rely on the fact if $\mathbf{NP} = \mathbf{P}$ then $\mathbf{P} = \mathbf{coNP}$ as proven in the base case. □

Before moving on we define another important complexity class that contains the **PH** and is relevant to the topic discussed in this thesis.

Definition 2.1.14
The class **PSPACE** is the class of problems solvable by a deterministic Turing machine within polynomial space.

Analogously the class **NPSPACE** is the class of problems solvable by a nondeterministic Turing machine within polynomial space. $\triangleleft$

Walter Savitch proved in [Sav70] that if a non-deterministic Turing machine can solve a problem within $s(n)$ space, then a deterministic Turing machine can solve the same problem within $s(n)^2$ space, which implies that **PSPACE** = **NPSPACE**.

It is known that **PH** $\subseteq$ **PSPACE**, however it is believed but still not proven that this is a strict containment. Otherwise if **PH** = **PSPACE** then every **PSAPCE**-complete problem L is also **PH**-complete, moreover it must be in some **PH** level, thus if $L \in \mathbf{\Sigma^p_k}$, then **PH** will collapse to $\mathbf{\Sigma^p_k}$.

Oracle

To gain a comprehensive overview of results presented in this thesis, we introduce, informally, the notion of an oracle.

An *oracle* is thought of as a black box that can solve a decision problem with one computational step. An *oracle machine* is a Turing machine with access to an oracle.

Papadimitriou in [Pap94] writes:
"If C is any deterministic or non-deterministic time complexity class, we can define C^A to be the class of all languages decided (or accepted) by machines of the same sort and time bound as C, that have now oracle A."

This means, for example, the class $\mathbf{P}^{\mathrm{SAT}}$ are all the problems decidable by a deterministic polynomial-time Turing machine with an access to an oracle that solves SAT (can check for satisfiability in one computational step).

If the problem that the oracle solves is complete for some class then we can use the name of the class in the notation, instead of the name of the problem. For example the above class is the same as $\mathbf{P}^{\mathbf{NP}}$, meaning the class of problems solvable by a deterministic polynomial Turing machine with an access to an oracle that solves some **NP**-complete problem.

2.2 Circuits

Intuitively a Boolean circuit is a diagram that shows how one can get an output from an encoded input with the use of Boolean operations like conjunction, disjunction and negation. The mathematical simplicity of Boolean circuits (relative to Turing machines) gives the impression that proving lower bounds for circuits is easier than those for Turing machines, which might eventually lead us to the proof of $\mathbf{P} \neq \mathbf{NP}$.

However, to prove a lower bound means to rule out a smaller size or depth for all circuits, which is why results on general circuits are mostly weak and rare. This made researchers focus their efforts on restricted classes of circuits like bounded depth circuits. It was, for example, proven that the TSP problem (an **NP**-complete problem) is not in $\mathbf{AC^0}$.

We refer the reader to [AB09] for more details on the lower bounds for circuits, and the issues this field of study faces as well as the corresponding open problems.

We mention in what follows the definitions and results which are important for this thesis.

We call a Boolean function with one output a *Boolean connective*. A set of Boolean connectives is called a *basis* and denoted by Ω. A common example is the *de Morgan* basis $\Omega = \{0, 1, \neg, \vee, \wedge\}$.

Definition 2.2.1
A *Boolean circuit* with input variables $x_1, x_2, \dots, x_n$, output variables $y_1, y_2, \dots, y_m$ and the basis of connectives $\Omega = \{g_1, g_2, \dots, g_k\}$ is a labeled acyclic directed graph, in which nodes with out-degree 0 are labeled by y_i's, nodes with in-degree 0 are labeled by x_i's or by constants from Ω, and nodes with in-degree $a \geq 1$ are labeled by functions from Ω of arity a.

A *Boolean formula* is a Boolean circuit in which every node has out-degree at most 1. $\triangleleft$

Recall that the in-degree (resp. out-degree) of a node in a directed graph is the number of edges incoming to it (resp. outgoing from it).

The nodes of a circuit are called gates. Most common ones are the AND, OR and NOT gates. Intuitively those gates behave like the corresponding Boolean connectives ($\wedge, \vee$ and $\neg$).

Another important gate is the majority gate (MAJ) which returns 1 if and only if at least 50% of its input is 1.

Note that we can get the results of the MAJ gate from AND and OR gates, for example for three inputs
$$\text{MAJ}(A, B, C) = \text{OR}(\text{AND}(A, B), \text{AND}(B, C), \text{AND}(A, C)).$$

We can also get the OR gate from MAJ as follows
$$\text{OR}(A, B, C) = \text{MAJ}(\text{MAJ}(A, B), C),$$

hence the AND gate as well but with the NOT gate
$$\begin{aligned}\text{AND}(A, B, C) &= \text{NOT}(\ \text{OR}(\text{NOT}(A), \text{NOT}(B), \text{NOT}(C))\ )\\ &= \text{NOT}(\ \text{MAJ}(\text{MAJ}(\text{NOT}(A), \text{NOT}(B)), \text{NOT}(C))\ ).\end{aligned}$$

Definition 2.2.2
Let C be a circuit.

- The *size of* C is the number of its gates, excluding the NOT gates.
- The *depth of* C is the maximum length of a directed path in the circuit C.

For a Boolean function f,

- $size_\Omega(f)$ denotes the minimal size of a circuit with basis Ω computing f, and
- $depth_\Omega(f)$ denotes the minimal depth of a circuit with basis Ω computing f. $\triangleleft$

We usually omit the index Ω when we use the *de Morgan* basis.

Definition 2.2.3
The *fan-in* of a connective $g \in \Omega$ is the number of inputs that g takes. Common choices are fan-in of two (bounded), or unbounded (meaning g can take any number of inputs). $\triangleleft$

The following theorem motivates the study of circuit complexity for Boolean functions, and will be useful for us in Chapter 4.

For any language $Z \subset \{0,1\}^*$, we define the Boolean function $Z_n : \{0,1\}^n \mapsto \{0,1\}$ to be the characteristic function of $Z \cap \{0,1\}^n$.

Theorem 2.2.1. *Let $Z \subseteq \{0,1\}^*$ be a language recognizable in polynomial time, i.e., $Z \in \mathbf{P}$. Then there exist a polynomial $p(\cdot)$ and a sequence of circuits $\{C_n : n \in N\}$ in de Morgan basis with one output, such that for all n, the following holds.*

1. *Z_n is computed by C_n.*
2. *The size of C_n is at most $p(n)$, i.e., $size(Z_n) \leq p(n)$ for all n.*

2.2.1 Circuit Classes

There are many circuit complexity classes, and they are usually defined as families of classes. We introduce here the general definition of the most used ones, and then in more detail the three classes we are interested in.

- **P\poly** is the class of languages computed by polynomial-size circuits.
- $\mathbf{NC^k}$ is the class of languages computed in $O(log^k n)$ depth, polynomial-size circuits, with bounded fan-in AND and OR gates and NOT gates.
- $\mathbf{AC^k}$ is the class of languages computed in $O(log^k n)$ depth, polynomial-size circuits, with unbounded fan-in AND and OR gates and NOT gates.

- $\mathbf{TC^k}$ is the class of languages computed in $O(log^k n)$ depth, polynomial-size circuits, with MAJ and NOT gates.

We have the following containments:

$$\mathbf{NC^0} \subseteq \mathbf{AC^0} \subseteq \mathbf{TC^0} \subseteq \mathbf{NC^1} \subseteq \mathbf{AC^1} \subseteq \mathbf{TC^1} \subseteq \mathbf{NC^2} \subseteq \mathbf{AC^2} \subseteq \mathbf{TC^2} \subseteq \cdots \subseteq \mathbf{P \backslash poly}.$$

We note that when $k = 0$ then we are speaking of circuits of constant depth $O(1)$ (because $log^0 n = 1$).

Theorem 2.2.2 (Karp and Lipton 1982)**.** *Assume that every* **NP** *language can be computed by a family of polynomial size circuits, that is* $\mathbf{NP} \subseteq \mathbf{P \backslash poly}$*.*

Then the polynomial hierarchy collapses to its second level, i.e., $\mathbf{PH} = \mathbf{\Sigma_2^p} = \mathbf{\Pi_2^p}$*.*

The relevant classes for us here are $\mathbf{NC^1}, \mathbf{TC^0}$ and $\mathbf{AC^0}$ which are defined in more details as follows.

Definition 2.2.4
$\mathbf{NC^1}$ is the class of languages L for which there exist circuit families $\{C_n : n \in N\}$ where each circuit C_n

- computes the characteristic function of L on inputs of length n,
- consists of AND and OR gates of fan-in two, and NOT gates, and
- has depth $O(log\ n)$ (and consequently has size $n^{O(1)}$). ◁

Definition 2.2.5
$\mathbf{TC^0}$ is the class of languages L for which there exist circuit families $\{C_n : n \in N\}$ where each circuit C_n

- computes the characteristic function of L on inputs of length n,
- consists of MAJ gates (with no bound on the fan-in), and NOT gates,
- has depth $O(1)$, and
- has size $n^{O(1)}$. ◁

Definition 2.2.6
$\mathbf{AC^0}$ is the class of languages L for which there exist circuit families $\{C_n : n \in N\}$ where each circuit C_n

- computes the characteristic function of L on inputs of length n,
- consists of AND and OR gates (with no bound on the fan-in), and NOT gates,
- has depth $O(1)$, and
- has size $n^{O(1)}$. ◁

Uniformity

In the definitions above, the complexity classes are defined in terms of non-uniform circuit families, that is, the existence of a circuit family is enough regardless of whether we can actually construct it.

However we shall focus on circuits which can be efficiently constructed, or what is known as uniform circuits. Throughout this thesis we always work with **DLogtime**-uniform circuits, defined as follows.

Definition 2.2.7 (**DLogtime**-uniform)
A circuit family $\{C_n\}$ is **DLogtime**-uniform if there exists a deterministic Turing machine that accepts it in $O(\log n)$ time ◁

The following are additional facts about $\mathbf{NC^1}$, which will be useful in Chapter 6 when describing the $\mathbf{NC^1}$ algorithm.

Barrington, Immerman, and Straubing proved in [MBIS90] that **DLogtime**-uniform $\mathbf{NC^1}$ is actually equivalent to **ALogtime** (the class of functions computable by an alternating Turing machine in logarithmic time). This result is significant because Samuel Buss in [Bus87] showed that there is an **ALogtime** algorithm for evaluating a propositional formula given a truth assignment.
Every $\mathbf{NC^1}$ predicate is computed by a **DLogtime**-uniform family of polynomial-size propositional formulas. A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is in $\mathbf{NC^1}$ if and only if $|f(x)| = |x|^{O(1)}$ (such that $|\cdot|$ denotes the string length) and the predicate $R_f(x,i,c)$, which denotes that the ith bit of $f(x)$ is c, is also in $\mathbf{NC^1}$ ([Mor05] and [CM05]).

Characterizing the classes $\mathbf{AC^0}$ and $\mathbf{TC^0}$

Based on [MBIS90] and [Imm99], Morioka laid out in [Mor05] an elegant first-order characterization of the classes $\mathbf{AC^0}$ and $\mathbf{TC^0}$.
Consider the first-order language $\mathcal{L}_{FO} = \{0, 1, n, +, \cdot, \leq, Bit\}$, such that $0, 1$, and n are constants, $+$ and $\cdot$ are binary functions, and $\leq$ and Bit are predicates.

We call first-order formulas over $\mathcal{L}_{FO}$ FO-sentences, for which the following holds.

- The interpretation of an FO-sentence φ is a binary string $I \in \{0,1\}^*$.
- n is a natural number that denotes the size of I.
- The function symbols $+, \cdot$, and the predicate $\leq$ have the usual meaning.
- $Bit(i)$ evaluates to true if and only if the ith bit of I is 1.
- The domain of the variables is $\{1, \dots, n\}$ (i.e., the set of indices of the string I).

An FO-sentence φ defines a relation $R \subseteq \{0,1\}^*$ such that $I \in R$ if and only if $I \vDash \varphi$ (φ evaluates to true under I), and we say that φ represents R.

For example the FO-sentence $\varphi = \neg Bit(n)$ is the sentence defining the relation R that represents binary numbers ending with 0 (even numbers), e.g., $I = (0, 1, 0) \vDash \varphi$ thus $I \in R$.

By introducing the majority quantifier M, we get FOM-sentences, such that a sentence $Mx\varphi(x)$ evaluates to true if and only if $\varphi(x)$ is true for at least half of the possible values for x.

We have the following two theorems

Theorem 2.2.3 (cf. [MBIS90, Imm99, Mor05])**.** *A relation R is in AC^0 if and only if it is representable by an FO-sentence.*

Theorem 2.2.4 (cf. [MBIS90, Imm99, Mor05])**.** *A relation R is in TC^0 if and only if it is representable by an FOM-sentence.*

2.3 Local Search Problems

In Chapter 6 we come across the fact that some witnessing problems are in fact **PLS**-complete. We present in what follows an introduction to the class **PLS**.

While complexity theory focuses mainly on decision problems, there are problems that are better represented as search problems.

We often can reduce a search problem to its decision counterpart in polynomial time, for which reason it is very common to study the decision counterpart instead of studying the search problem itself. However this approach is not optimal for many reasons, Krentel showed in [Kre88] that the traveling salesperson search problem is harder than the clique search problem, even though the decision counterparts of both problems are **NP**-complete. This implies that the complexity of search problems depends on properties that are lost in the translation to decision problems.

Another reason for studying search problems is the fact that it is not always possible to find a polynomially equal decision problem. It was demonstrated in [BCE$^+$98], that search problems that are complete for certain complexity classes like **PLS** (Polynomial Local Search) are not polynomially equivalent to any decision problem.

We first define search problems and local search problems.

Definition 2.3.1 (Search Problem)
A search problem is a relation R over $\{0,1\}^* \times \{0,1\}^*$.

An algorithm solves a search problem R if, when given an $x \in \{0,1\}^*$, it either returns a solution s such that $(x, s) \in R$ or reports (correctly) that no such s exists. $\triangleleft$

Definition 2.3.2 (Local Search Problem)
A local search problem L can be either a maximization or a minimization problem, and has a set of instances $D_L \subseteq \{0,1\}^*$.

For each instance $x \in D_L$, there is a finite set of solutions $F_L(x) \subseteq \{0,1\}^*$.

For each instance $x \in D_L$, and a solution $s \in F_L(x)$, there is

- a cost $c_L(x,s) \in \mathbb{Z}^+$, and
- a set of neighboring solutions $N(x,s) \subseteq F_L(x)$, based on a predefined neighborhood relation. A solution s is locally optimal if it is the optimal solution in its neighborhood.

An algorithm solves a local search problem L if when given an $x \in \{0,1\}^*$, it either returns a solution s such that $(x,s) \in R_L$ or reports (correctly) that no such s exists, where $R_L = \{(x,s) : x \in D_L,\ s \in F_L(x) \text{ and } s \text{ is locally optimal}\}$. $\triangleleft$

Example 2.1
Recall that a Boolean formula is in 3-CNF if it is of the form $C_1 \wedge \ldots \wedge C_k$ such that each C_i is a disjunction of at most three literals.

Let us consider the 3-SAT problem.

Problem 3-SAT	
Input:	A Boolean formula φ in 3-CNF.
Question:	Does there exist a satisfying assignment for φ?

We can describe 3-SAT as a local search problem as follows. "Given a Boolean formula φ in 3-CNF, find an assignment that satisfies the maximum number of clauses in φ".

Then we have the following.

- The set of instances are Boolean formulas in 3-CNF.
- For an instance φ, a solution $s \in F_L(\varphi)$ is a string of 1s and 0s to denote an assignment of the formula's literals, e.g., if φ has just four literals l_1, l_2, l_3, l_4 then $s = 0000$ means assign 0 to all literals, and $s = 0100$ means assign 1 to l_2 and 0 to the rest of the literals. $F_L(\varphi)$ is the set of all possible assignments.
- The cost $c_L(\varphi, s)$ of a solution s is the number of satisfied clauses in φ.
- The neighborhood of a solution is defined as follows.
  $N(\varphi, s) = \{s' | s' \in F_L(\varphi) \text{ and } hdist(s,s') = 1\}$, where $hdist(s,s')$ is the Hamming distance between s and s', (all solutions which differ from s by one literal assignment). For instance, again, if φ has just four literals and $s = 0000$, then the neighborhood of s is the set $\{0001, 0010, 0100, 1000\}$.

- A solution s is locally optimal if it has the maximal cost in its neighborhood.

Note that if φ is satisfiable then the algorithm that solves the local search problem will find a satisfying assignment, since it will be the solution with the maximum cost. If φ is not satisfiable, then the algorithm will find the solutions which satisfy the maximum number of clauses within their neighborhoods.

Example 2.2
Consider for example the 3-CNF instance.

$$\varphi = (\neg y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y) \wedge (\neg y \vee z) \wedge (\neg x \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee y \vee \neg z)$$

φ has three literals and 8 clauses, and it is not satisfiable. The algorithm, however, will find two locally optimal solutions, or rather assignments for the literals xyz in this exact order, namely $s_1 = 111$ such that $N(\varphi, s_1) = \{110, 101, 011\}$, and $s_2 = 000$ such that $N(\varphi, s_2) = \{001, 010, 100\}$. Each of s_1 and s_2 satisfies 7 clauses, which is the maximum number of satisfied clauses in the respective neighborhood. In the table below we can see the satisfied clauses by each possible assignment.

Assignment	# Sat	$\neg y$	$x \vee \neg y$	$\neg x \vee y$	$x \vee y$	$\neg y \vee z$	$\neg x \vee z$	$\neg x \vee y \vee \neg z$	$x \vee y \vee \neg z$
111	7		✓	✓	✓	✓	✓	✓	✓
011	6			✓	✓	✓	✓	✓	✓
101	6	✓	✓		✓	✓	✓		✓
001	6	✓	✓	✓		✓	✓	✓	
110	5		✓	✓	✓			✓	✓
010	5			✓	✓		✓	✓	✓
100	6	✓	✓		✓	✓		✓	✓
000	7	✓	✓	✓		✓	✓	✓	✓

Analogously to **P** and **NP** for decision problems, we define the classes **FP** and **FNP** for search problems.

Definition 2.3.3 (The Class **FNP** (a.k.a $\mathbf{NP_s}$))
A search problem R is in **FNP** if

- for any $(x, s) \in R$, $|s|$ is polynomially bounded in $|x|$, and
- there is an algorithm that, given the pair (x, s), can verify in polynomial time if $(x, s) \in R$. ◁

Note that this definition is equivalent to saying: A search problem is in **FNP** if it can be solved in polynomial time with a non-deterministic Turing machine.

Definition 2.3.4 (The Class **FP** (a.k.a $\mathbf{P_s}$))
A search problem is in **FP** if it can be solved in polynomial time with a deterministic Turing machine. ◁

These definitions imply the following result.

Lemma 2.3.1 ([JPY85])**.**

$$\mathbf{FP} = \mathbf{FNP} \textit{ if and only if } \mathbf{P} = \mathbf{NP}.$$

Definition 2.3.5 (The class **PLS**)
A local search problem L is in the class of Polynomial Local Search (PLS) problems if the following conditions hold :

- The set of instances D_L is polynomial time recognizable, i.e., for any x we can check if $x \in D_L$ in polynomial time,
- each instance $x \in D_L$ has a finite set of solutions, in which
- each solution $s \in F_L(x)$ has a polynomially bounded length $p(|x|)$, and
- there exist three polynomial time algorithms which are as follows.
    - A_L: Returns a solution s from $F_L(x)$, given instance x.
    - B_L: Checks if $s \in F_L(x)$, and if it is, computes the cost of s on x.
    - C_L: Given x, s, based on the cost, chooses the optimal solution in the neighborhood (local optimum) of s.

For simplicity, **PLS** will be used to denote the class of all relations R_L, arising from **PLS**-problems. ◁

We observe that the definition implicitly defines a *standard* local search algorithm: Given x, use A_L to produce s (an initial solution), then keep applying B_L and C_L until a locally optimal solution is found.

Since the set of solutions is finite, we know that the described algorithm will halt, but the question is how long will it take to find the optimal solution? The algorithm itself might take up to exponential time, however other means might be faster.

This gives rise to what is known as the *standard algorithm problem*:

"Given x, find the local optimum s that would be the output of the standard local search algorithm for L on input x."

Johnson, Papadimitriou, and Yannakakis in [JPY85], introduced a couple of lemmas on how **PLS** is related to other complexity classes.

Lemma 2.3.2 ([JPY85])**.** *There is a* **PLS** *problem L whose standard algorithm problem is* **NP***-hard.*

Proof. Let L be a local search problem with the same instances as SAT (an **NP**-hard problem) such that the following holds.

1. The solutions for an instance x with n variables are the truth assignments to those variables. Truth assignments are represented as elements of $\{0, 1\}^n$, hence the solutions represent integers from 0 to $2^n - 1$.

2. The neighborhood of a solution $s > 0$ is simply $N(x, s) = \{s-1\}$, the neighborhood of 0 is empty.
3. The cost $C_L(x, s)$ is 0 if s represents a satisfying assignment for x, otherwise the cost is the integer s.
4. The initial solution returned by the algorithm A_L is simply $2^n - 1$ (the assignment where all the variables are set to true) and the goal is to minimize the cost.

The description of L implies that

- we can check in polynomial time, for any x, if x is indeed a Boolean formula (i.e., $x \in D_L$),
- each instance x has a finite set of solutions, and the size of each solution s is bounded by the number of variables n in the instance x, and
- the three polynomial-time algorithms A_L, B_L, and C_L, described in the definition of **PLS**, exist.

Thus by the definition of the class **PLS**, L is indeed a **PLS** problem.

For an instance x of SAT, we assume, without loss of generality, that the integer 0 does not represent a satisfying assignment for x. Based on the description above, the standard local search algorithm for L will output 0 if and only if x is not satisfiable. Thus the standard algorithm problem for L is **NP**-hard. □

Lemma 2.3.3 ([JPY85]). **FP** $\subseteq$ **PLS** $\subseteq$ **FNP**.

Proof. Let $R \in$ **FP**. Then we can formalize it as a **PLS** problem by defining the three algorithms A_R, B_R and C_R as follows:

Let A_R be the polynomial-time algorithm that solves R (which exists by the definition of an **FP** problem).

We define $F_R(x) = \{y : (x, y) \in R\}$, the cost of y on x is 0 if $y \in F_R(x)$, and the neighborhood is always the empty set (thus making s a trivial local optimum), which means B_R will use the polynomial-time algorithm to recognize the membership of y, and C_R will always choose y as the optimal local solution. This mean $R \in$ **PLS** which entails that **FP** $\subseteq$ **PLS**.

On the other hand since the definition of **PLS** implies the existence of the algorithm C_L which can verify that s is the optimal solution for instance x, thus any $L \in$ **PLS** is also in **FNP** which implies that **PLS** $\subseteq$ **FNP**. □

Definition 2.3.6
Let P_1 and P_2 be two search problems. Then we say that P_1 is polynomial-time reducible to P_2 if there exists two polynomial-time functions f, g, such that

for each instance I_1 of P_1, $g(I_1)$ is an instance of P_2, and if s is a solution of $g(I_1)$ then $f(I_1, s)$ is a solution to I_1. ◁

Lemma 2.3.4. *If a* **PLS** *problem L is* **NP***-hard, then* **NP** = **coNP***.*

Proof. The proof is by Johnson, Papadimitriou, and Yannakakis published in [JPY85].

Let $L \in \mathbf{PLS}$ be an **NP**-hard problem, and X be some **NP** problem.

We shall think of the problem X as a special case of a search problem, where the possible solutions are singleton sets containing either 1 or 0 (yes or no).

Since L is **NP**-hard then X is polynomial time reducible to L, meaning there exists two polynomial time function f and g, such that,

for each I_x, an instance of X, $g(I_x)$ is an instance of L, and if s is a solution for $g(I_x)$ then $f(I_x, s)$ is a solution for I_x.

Since L is a **PLS** problem, we know that L always has a solution, (this is implied by its definition). This means that, for any I_0 a NO-instance of X, we get $g(I_0)$ an instance of L, we can find s_0 the solution of $g(I_0)$ with the algorithms A_L, B_L, C_L, and we will then have $f(I_0, s_0) = 0$.

With the above described procedure we can recognize all NO-instances of X non-deterministically and in polynomial time, which means that, for any problem X if $X \in \mathbf{NP}$ then $\overline{X} \in \mathbf{NP}$. This implies

$$\begin{aligned} A \in \mathbf{NP} &\Rightarrow \overline{A} \in \mathbf{NP} \\ &\Rightarrow A \in \mathbf{coNP} \\ &\Rightarrow \mathbf{NP} \subseteq \mathbf{coNP}, \text{ and} \\ B \in \mathbf{coNP} &\Rightarrow \overline{B} \in \mathbf{NP} \\ &\Rightarrow B \in \mathbf{NP} \\ &\Rightarrow \mathbf{coNP} \subseteq \mathbf{NP} \end{aligned}$$

which eventually implies that **NP** = **coNP**. □

The concept of an oracle machine, introduced earlier in the section of basic complexity theory, is also applicable in local search problems. Two classes which we will briefly encounter later in this thesis are the following.

The class $\mathbf{FP}^{\Sigma^p_{i-1}}$ is the class of search problems solvable in polynomial-time by a deterministic Turing machine with an access to an oracle that can solve a Σ^p_{i-1}-complete problem, and

the class $\mathbf{PLS}^{\Sigma^p_{i-1}}$ is the class of search problems satisfying the conditions in Definition 2.3.5 while having access to an oracle that can solve a Σ^p_{i-1}-complete problem

CHAPTER 3

Quantified Propositional Calculus

In this chapter we take a look at the size of a shortest proof of valid formulas in some proof system as a function of the size of the formula, Cook and Reckhow [CR79] have proved important results connecting proof size in propositional proofs to complexity theory, which was then generalized to similar results in quantified proofs by Cook and Morioka in [CM05].

We first start with some definitions and preliminaries.

Definition 3.0.1 (The Syntax of Quantified Boolean Formulas (QBFs))
Given a set of propositional Boolean variables $\mathcal{P}$. The set of QBFs is defined inductively as follows.

1. The constants $\bot$, $\top$, and every $p \in \mathcal{P}$ are QBFs.
2. If φ_1 and φ_2 are QBFs then $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ and $\neg\varphi_1$ are also QBFs.
3. If φ is a QBF then so are $\exists p\ \varphi$ and $\forall p\ \varphi$, where $p \in \mathcal{P}$.

We call the propositions $p \in \mathcal{P}$ atomic formulas. ◁

Definition 3.0.2 (The Semantics of Quantified Boolean Formulas (QBFs))
Let $\mathcal{I}$ be an interpretation represented as a set of atoms, such that an atom p is true under $\mathcal{I}$ if and only if $p \in \mathcal{I}$. Let $\varphi[x/t]$ denote the replacement of every free occurrence of x in φ with t.

Then we define the truth value $v_{\mathcal{I}}(\varphi)$ of a QBF inductively as follows:

- If $\varphi = \top$ then $v_{\mathcal{I}}(\varphi) = 1$.
- If $\varphi = \bot$ then $v_{\mathcal{I}}(\varphi) = 0$.
- If $\varphi = p \in \mathcal{P}$ then $v_{\mathcal{I}}(\varphi) = 1$ if $p \in \mathcal{I}$, otherwise $v_{\mathcal{I}}(\varphi) = 0$.

- If $\varphi = \neg\varphi_1$ then $v_{\mathcal{I}}(\varphi) = 1 - v_{\mathcal{I}}(\varphi_1)$.
- If $\varphi = \varphi_1 \wedge \varphi_2$ then $v_{\mathcal{I}}(\varphi) = min(\{v_{\mathcal{I}}(\varphi_1), v_{\mathcal{I}}(\varphi_2)\})$.
- If $\varphi = \varphi_1 \vee \varphi_2$ then $v_{\mathcal{I}}(\varphi) = max(\{v_{\mathcal{I}}(\varphi_1), v_{\mathcal{I}}(\varphi_2)\})$.
- If $\varphi = \forall p\ \varphi_1$ then $v_{\mathcal{I}}(\varphi) = v_{\mathcal{I}}(\varphi_1[p/\top] \wedge \varphi_1[p/\bot])$.
- If $\varphi = \exists p\ \varphi_1$ then $v_{\mathcal{I}}(\varphi) = v_{\mathcal{I}}(\varphi_1[p/\top] \vee \varphi_1[p/\bot])$.

We say that φ is true under $\mathcal{I}$ if and only if $v_{\mathcal{I}}(\varphi) = 1$. ◁

Note that to reason about the truth value of a QBF with free variables we need a specific interpretation $\mathcal{I}$ to evaluate those variables. However with closed QBFs (QBFs with no free variables) there is no need to refer to a particular interpretation.

Example 3.1
To see the importance of $\mathcal{I}$, let $\mathcal{I} = \{y\}$ and consider $\varphi(x, y) = x \wedge y$.

If $\varphi_1 = \exists x\varphi(x, y)$ then since $y \in \mathcal{I}$ and y is free in φ_1, we can replace each y with $\top$, and get $\varphi_1' = \exists x(x \wedge \top)$.

The formula φ_1' clearly evaluates to true, i.e., $v_{\mathcal{I}}(\varphi_1') = v_{\mathcal{I}}((\top \wedge \top) \vee (\bot \wedge \top)) = 1$.

If $\varphi_2 = \exists x\forall y\varphi(x, y)$, then φ_2 has no free variables, which implies that $\mathcal{I}$ is no longer helpful, and we have to consider all possible replacements for y.

In this case φ_2 will evaluate to false, i.e., $v_{\mathcal{I}}(\varphi_2) = v_{\mathcal{I}}(\forall y(\top \wedge y) \vee \forall y(\bot \wedge y)) = 0$.

Definition 3.0.3
Let $Q_i \in \{\forall, \exists\}$ and $p_i, q_i \in \mathcal{P}$. Then a quantified propositional formula φ is in *prenex normal form* (PNF) if

$$\varphi = Q_1 p_1 \cdots Q_n p_n \psi(\overline{p}, \overline{q})$$

and ψ is purely propositional, where $\overline{p}$ is a sequence of all quantified variables in φ and $\overline{q}$ is a sequence of all free variables in φ. ◁

Stockmeyer and Meyer proved in [SM73] that the problem of evaluating whether a prenex QBF is true, is a **PSPACE**-complete problem. This problem is often called QSAT and is a generalization of SAT.

Prenex QBFs have a hierarchy of their own, which is similar to the polynomial hierarchy and is defined as follows.

Definition 3.0.4 ([Kra95])
We define a hierarchy of QBFs in prenex normal form as follows.

- $\Sigma_0^q = \Pi_0^q$ denote the set of propositional formulas.
- The classes Σ_{i+1}^q and Π_{i+1}^q are the smallest classes satisfying

a) $\Sigma_i^q \cap \Pi_i^q \subseteq \Sigma_{i+1}^q \cup \Pi_{i+1}^q$,
b) both Σ_{i+1}^q and Π_{i+1}^q are closed under $\vee$ and $\wedge$,
c) if $\varphi \in \Sigma_{i+1}^q$ then $\neg\varphi \in \Pi_{i+1}^q$,
d) if $\varphi \in \Pi_{i+1}^q$ then $\neg\varphi \in \Sigma_{i+1}^q$,
e) Σ_{i+1}^q is closed under existential quantification,
f) Π_{i+1}^q is closed under universal quantification. $\triangleleft$

3.1 Quantified Proof Systems

In what follows we use the following notations:

- TAUT: This is the set of propositional tautologies.
- Σ^*: This is the set of all finite strings over the finite alphabet Σ.
- $\mathscr{L}$: This is the set of functions $f : \Sigma_1^* \longmapsto \Sigma_2^*$, where Σ_1, Σ_2 are any finite alphabets, such that f can be computed by a deterministic Turing machine in time bounded by a polynomial in the size of the input.
- For some string x, we use $|x|$ to denote the length of x, which is the total number of symbol occurrences in x, and if π is a proof, then $|\pi|$ would be the size of the proof π.

Definition 3.1.1
If $L \subset \Sigma^*$ then a *proof system* for L is the function $f : \Sigma_1^* \longmapsto L$ for some alphabet Σ_1 and f in $\mathscr{L}$ such that f is onto. $\triangleleft$

Definition 3.1.2
A proof system for L is *polynomially bounded* if and only if there is a polynomial $p(\cdot)$ such that $y = f(x)$ and $|x| \leq p(|y|)$.

When $y = f(x)$ we say that x is a proof of y, and if the condition $|x| \leq p(|y|)$ also holds, we say that x is a short proof of y. $\triangleleft$

Lemma 3.1.1. *[CR79]*

NP *is closed under complementation, i.e.,* **NP** = **coNP**, *if and only if* TAUT *is in* **NP**.

Proof.

NP is closed under complementation $\Rightarrow$ TAUT is in **NP**.

A propositional formula φ is satisfiable if and only if $\neg\varphi$ is not valid, i.e., not in TAUT. This means that SAT is the complement of TAUT. Since SAT is **NP**-complete, it is in **NP**. This implies that if **NP** is closed under complementation then TAUT is in **NP**.

TAUT is in **NP** $\Rightarrow$ **NP** is closed under complementation.

If TAUT is in **NP** then there is a non-deterministic procedure p that accepts tautologies in polynomial time.
Since SAT is **NP**-complete, then any $L \in$ **NP** is reducible to SAT, in the sense that for any instance $x \in L$ there exists a polynomial time function $f \in \mathscr{L}$ such that $x \in L$ if and only if $f(x) \in$ SAT.
As mentioned in the proof of the first direction, any propositional formula φ is satisfiable if and only if $\neg\varphi$ is not valid, thus there exist a polynomial-time function g, such that $y \in$ SAT if and only if $g(y) \notin$ TAUT.

We can define a non-deterministic procedure for accepting $\overline{L} \in$ **coNP** (the complement of L) as follows.
For any instance z, accept z to be in $\overline{L}$ if and only if the procedure p accepts $g(f(z))$. We can do this because

$$\begin{aligned} z \in \overline{L} &\text{ if and only if } z \notin L \\ &\text{ if and only if } f(z) \notin \text{SAT} \\ &\text{ if and only if } g(f(z)) \in \text{TAUT}. \end{aligned}$$

This entails that **NP** is closed under complementation. □

Lemma 3.1.2. *[CR79]*

A set L is in **NP** *if and only if $L = \emptyset$ or L has a polynomially bounded proof system.*

Proof.

If L is in **NP** then $L = \emptyset$ or L has a polynomially bounded proof system.

If L is in **NP** then there exists a non-deterministic Turing machine M that accepts L in polynomial time.
If $L = \emptyset$ then the proof is done. Otherwise if $L \neq \emptyset$ then we can define a function f as follows: If the input x codes a computation of M that accepts y then $f(x) = y$, and if not then $f(x) = y_0$ for some fixed $y_0 \in L$. Since y (when accepted) is in $L \in$ **NP**, then by definition the computation x of the Turing machine M that accepts y is bounded by a polynomial in the size of y, meaning $|x| \leq p(|y|)$, which implies that f is a polynomially bounded proof system for L.

If $L = \emptyset$ or L has a polynomially bounded proof system then L is in **NP**.

If $L = \emptyset$ then the Turing machine that rejects any input represents L, thus $L \in$ **NP**.

On the other hand if f is a polynomially bounded proof system for L, then we know that for any input x, $f(x) = y$ if and only if $y \in L$ and $|x| \leq p(|y|)$ for some polynomial $p(\cdot)$. So we can define a non-deterministic procedure for accepting L as follows. For input y guess a short proof x of y and then verify $f(x) = y$. This procedure implies that L is in **NP**. □

By combining the proofs of Lemma 3.1.1 and Lemma 3.1.2 we get a proof for the following

Theorem 3.1.3. *There exists a propositional proof system in which every tautology has a polynomial size proof if and only if the class* $\mathbf{NP}$ *is closed under complementation, i.e.,* $\mathbf{NP} = \mathbf{coNP}$ *holds.*

The generalisation of these results to quantified propositional proofs have been introduced by Cook and Morioka in [CM05].

Definition 3.1.3 (Quantified Propositional Proof System)
A *quantified propositional Proof System* is a proof system $Q : \Sigma_1^* \longmapsto L$, where $\Sigma_1 = \{0, 1\}$ and L is the set of valid QBFs. $\triangleleft$

Theorem 3.1.4. (i) *There exists a quantified propositional proof system Q in which every valid QBF φ has a proof of size polynomial in $|\varphi|$ if and only if* $\mathbf{NP} = \mathbf{PSPACE}$.

(ii) *For every $i \geq 0$, there exists a quantified propositional proof system Q in which every valid Σ_i^q-formula φ has a proof of size polynomial in $|\varphi|$ if and only if* $\mathbf{NP} = \mathbf{\Pi^P_{i+1}}$.

Proof.

(i) Let L be any **PSPACE** problem, and let Q be a quantified propositional proof system, in which every valid QBF formula φ has a proof of size polynomial in $|\varphi|$. By definition Q can be computed by a deterministic Turing machine in polynomial-time. Let M_Q be this Turing machine.

Then we can define a non-deterministic procedure that accepts any $L \in \mathbf{PSPACE}$ in polynomial time as follows. Reduce L to a QBF, say φ_L, and then accept L if and only if M_Q accepts that φ_L is valid. This implies that $L \in \mathbf{NP}$, thus $\mathbf{PSPACE} \subseteq \mathbf{NP}$, and since we already know that $\mathbf{NP} \subseteq \mathbf{PSPACE}$, we get $\mathbf{NP} = \mathbf{PSPACE}$.

On the other hand if $\mathbf{NP} = \mathbf{PSPACE}$ then by Lemma 3.1.2 any $L \in \mathbf{PSPACE}$ has a polynomially bounded proof system.

(ii) Similar to the proof of (i). However, we rely on the fact, proven in [Wra76], that evaluating a Π^q_{i+1}-formula is $\mathbf{\Pi^P_{i+1}}$-complete, and dually evaluating a Σ^q_{i+1}-formula is $\mathbf{\Sigma^P_{i+1}}$-complete, for any $i \geq 0$. $\square$

3.2 Gentzen-style Sequent Calculus for QBF

The basic objects of sequent calculi are sequents, i.e., ordered pairs of finite (possibly empty) sequences of formulas written as

$$\varphi_1, \varphi_2, \ldots, \varphi_l \longrightarrow \psi_1, \psi_2, \ldots, \psi_k.$$

The intuitive meaning of such a sequent is that the conjunction of the φ_i's implies the disjunction of the ψ_i's, i.e., $\bigwedge_{i=1}^{l} \varphi_i \supset \bigvee_{i=1}^{k} \psi_i$.

We use capital Greek letters (e.g., Γ, Δ, Π) to denote finite sequences of formulas

3.2.1 Propositional Sequent Calculus (PK)

PK has the following three axioms.

$\varphi \longrightarrow \varphi$ where φ is an atom, $\quad \bot \longrightarrow$, $\quad \longrightarrow \top$

In the following inference rules, we use the letters r and l (meaning right and left) to denote the side at which the rule is applied.

Weakening Rules:

$$[\mathrm{Wl}]\ \frac{\Gamma \longrightarrow \Delta}{\varphi, \Gamma \longrightarrow \Delta} \qquad [\mathrm{Wr}]\ \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \varphi}$$

Exchange Rules:

$$[\mathrm{El}]\ \frac{\Gamma_1, \varphi, \psi, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, \psi, \varphi, \Gamma_2 \longrightarrow \Delta} \qquad [\mathrm{Er}]\ \frac{\Gamma \longrightarrow \Delta_1, \varphi, \psi, \Delta_2}{\Gamma \longrightarrow \Delta_1, \psi, \varphi, \Delta_2}$$

Contraction Rules:

$$[\mathrm{Cl}]\ \frac{\varphi, \varphi, \Gamma \longrightarrow \Delta}{\varphi, \Gamma \longrightarrow \Delta} \qquad [\mathrm{Cr}]\ \frac{\Gamma \longrightarrow \Delta, \varphi, \varphi}{\Gamma \longrightarrow \Delta, \varphi}$$

The Cut Rule:

$$[\mathrm{cut}]\ \frac{\Gamma \longrightarrow \Delta, \varphi \qquad \varphi, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}$$

Logical Rules:

- Negation ($\neg$):

$$[\neg \mathrm{l}]\ \frac{\Gamma \longrightarrow \Delta, \varphi}{\neg\varphi, \Gamma \longrightarrow \Delta} \qquad [\neg \mathrm{r}]\ \frac{\varphi, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \neg\varphi}$$

- Conjunction ($\wedge$):

$$[\wedge \mathrm{l1}]\ \frac{\varphi, \Gamma \longrightarrow \Delta}{\varphi \wedge \psi, \Gamma \longrightarrow \Delta} \qquad [\wedge \mathrm{l2}]\ \frac{\psi, \Gamma \longrightarrow \Delta}{\varphi \wedge \psi, \Gamma \longrightarrow \Delta}$$

$$[\wedge r]\ \frac{\Gamma \longrightarrow \Delta, \varphi \qquad \Gamma \longrightarrow \Delta, \psi}{\Gamma \longrightarrow \Delta, \varphi \wedge \psi}$$

- Disjunction ($\vee$):

$$[\vee l]\ \frac{\varphi, \Gamma \longrightarrow \Delta \qquad \psi, \Gamma \longrightarrow \Delta}{\varphi \vee \psi, \Gamma \longrightarrow \Delta}$$

$$[\vee r1]\ \frac{\Gamma \longrightarrow \Delta, \varphi}{\Gamma \longrightarrow \Delta, \varphi \vee \psi} \qquad [\vee r2]\ \frac{\Gamma \longrightarrow \Delta, \psi}{\Gamma \longrightarrow \Delta, \varphi \vee \psi}$$

For convenience we will extend the system by two rules introducing the implication. Note that to avoid confusion between implication and sequent arrows we use the symbol $\supset$ to denote implication.

- Implication ($\supset$):

$$[\supset l]\ \frac{\Gamma \longrightarrow \Delta, \varphi \qquad \psi, \Gamma \longrightarrow \Delta}{\varphi \supset \psi, \Gamma \longrightarrow \Delta} \qquad [\supset r]\ \frac{\varphi, \Gamma \longrightarrow \Delta, \psi}{\Gamma \longrightarrow \Delta, \varphi \supset \psi}$$

Recall that $\varphi \supset \psi$ is equivalent to $\neg\varphi \vee \psi$, which means that these two rules abbreviate the derivation of $\neg\varphi \vee \psi$ (left and right) which can be done by the right application of the introduction of negation and disjunction as well as the contraction rule.

We define some important notions and notations.

- The formulas Γ, Δ are called *side formulas* or the *context*.
- The *auxiliary formulas* are those which appear in the upper sequent and are not side formulas, i.e., premise formulas (e.g., φ in $\wedge$r).
- The *principal formulas* are those which appear in the lower sequent and are not side formulas, i.e., conclusion formulas (e.g., $\varphi \wedge \psi$ in $\wedge$r).
- A PK-proof is a sequence of sequents, in which every sequent is either an axiom or is derived from previous sequents by one of the inference rules.
- The last sequent in a proof is called the end-sequent.
- If s and t are two inference steps in a proof π, we say that s *precedes* t if s occurs in a subproof of π ending with t.
- The *successor* of a formula φ in a proof is defined as follows.
  - If φ is a formula of Γ, then the successor of φ is the first occurrence (starting from left) of φ in the left side of the lower sequent.
  - If φ is a formula of Δ, then the successor of φ is the first occurrence (starting from left) of φ in the right side of the lower sequent.

  - If φ is not a side formula, then the principal formula of a rule is the successor of φ.

- The concept of principal formulas and successor is not defined for the cut rule.

Definition 3.2.1 (Polynomial-size PK-proofs)
A family of sequents has *polynomial-size* PK-*proofs* if there exists a polynomial $p(\cdot)$ such that, for every formula φ in the family, there exists a PK-proof of φ whose size is at most $p(|\varphi|)$. ◁

Definition 3.2.2 (Treelike proofs)
A proof where each sequent occurs as an upper sequent of an inference step at most once, is *a treelike proof*. ◁

Definition 3.2.3 (p-simulation)
Let Q_1 and Q_2 be quantified propositional proof systems. We say that Q_2 *p-simulates* Q_1 if and only if there exists a polynomial time function h such that if π_1 is a Q_1-proof of φ, then $h(\pi_1)$ is a Q_2-proof of φ.
We say that Q_1 and Q_2 are p-equivalent if they p-simulate each other. ◁

3.2.2 Sequent Calculi for QBF

Krajíček and Pudlák introduced in [KP90] Gentzen-like sequent calculi for quantified Boolean formulas. Those systems were denoted as KPG, KPG$_i$ and KPG$_i^*$ in [CM05] (we adopt these names here). Cook and Morioka introduced the systems G, G_i and G_i^* by modifying the definition of KPG systems to have a better correspondence between the bounded arithmetic theories T_i^2 and S_i^2 and the defined calculus. We describe the different proof systems in the following.

Definition 3.2.4 (KPG)
We obtain the system KPG by extending PK with the following rules:

$$[\exists\mathrm{l}]\ \frac{\varphi(p), \Gamma \longrightarrow \Delta}{\exists x\varphi(x), \Gamma \longrightarrow \Delta} \qquad [\exists\mathrm{r}]\ \frac{\Gamma \longrightarrow \Delta, \varphi(\psi)}{\Gamma \longrightarrow \Delta, \exists x\varphi(x)}$$

$$[\forall\mathrm{l}]\ \frac{\varphi(\psi), \Gamma \longrightarrow \Delta}{\forall x\varphi(x), \Gamma \longrightarrow \Delta} \qquad [\forall\mathrm{r}]\ \frac{\Gamma \longrightarrow \Delta, \varphi(p)}{\Gamma \longrightarrow \Delta, \forall x\varphi(x)}$$

Conditions for quantifier rule applications are as follows.

- The variable p in $\exists$l and $\forall$r does not occur free in the lower sequents, and is called an eigenvariable.

- ψ can be any proper formula free for substitution for x in φ, i.e., no free variable in ψ should be bounded in $\varphi(\psi)$. We call ψ the target of the corresponding inference.

- Additionally, in the axiom $\varphi \longrightarrow \varphi$, φ is allowed to be any QBF. $\triangleleft$

Definition 3.2.5 (KPG$_i$ and KPG$_i^*$)
For $i \geq 1$, KPG$_i$ is obtained from KPG by requiring that all formulas in the proof to be in $\Sigma_i^q \cup \Pi_i^q$. These are the formulas that are in prenex form with at most $i-1$ quantifier alternations.

KPG$_i^*$ is KPG$_i$ restricted to treelike proofs. $\triangleleft$

Definition 3.2.6 (G, G_i and G_i^*)
G is obtained by augmenting PK with the four quantifier-introduction rules from definition 3.2.4, with the additional restriction that the target ψ of every ∀l and ∃r step is quantifier-free.

For $i \geq 0$, G_i is G with cuts restricted to $\Sigma_i^q \cup \Pi_i^q$-formulas. G_i^* is the treelike version of G_i. $\triangleleft$

Definition 3.2.7
Parameter variables are the free variables in a G-proof which occur in the end-sequent. $\triangleleft$

Note that restricting KPG$_i$ to reason only with formulas in $\Sigma_i^q \cup \Pi_i^q$ means that KPG$_i$ can not prove all true QBFs, i.e., it is not complete for any fixed i. However the modification introduced by Cook and Morioka in [CM05] gives us complete proof systems G_i which have a more natural correspondence with the theories of bounded arithmetic T_2^i and S_2^i, which will be introduced in the next chapter.
For any $i \geq 0$, G_{i+1} is stronger than G_i, while both can reason about any QBF, G_{i+1}-proofs might be shorter, because it allows more complex cut formulas.

We use in the following proof the notion of a subformula, and we shall adopt the definition by Takeuti in [Tak87] (Definition 6.1 on p.28).

Definition 3.2.8
The set of *immediate subformulas* of a formula φ is defined as follows.

- If φ is an atomic formula, then it has no immediate subformulas.
- If φ is $\neg\varphi_1$, then the set of immediate subformulas is $\{\varphi_1\}$.
- If φ is $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2$, or $\varphi_1 \supset \varphi_2$, then the set of immediate subformulas is $\{\varphi_1, \varphi_2\}$.
- If φ is $\forall x\ \varphi_1(x)$ or $\exists x\ \varphi_1(x)$, then the set of immediate subformulas is $\{\varphi_1(x)\}$.

The set of *subformulas* of φ is the reflexive transitive closure of the set of immediate subformulas. $\triangleleft$

Note that we only need here the immediate subformulas and we mention the definition of proper subformulas for completeness.

Lemma 3.2.1. *Let $\exists x\varphi(x)$ and ψ be QBFs, and let $\varphi(\psi)$ be the result of substituting ψ for all free occurrences of x in $\varphi(x)$, where ψ satisfies the restriction for the target formula in $\forall l$ and $\exists r$. The following four sequents have cut-free G_0^*-proofs of size $O(|\varphi(\psi)|^2)$:*

(T1): $\psi, \varphi(\psi) \longrightarrow \varphi(\top)$

(T2): $\varphi(\psi) \longrightarrow \varphi(\bot), \psi$

(T3): $\psi, \varphi(\top) \longrightarrow \varphi(\psi)$

(T4): $\varphi(\bot) \longrightarrow \psi, \varphi(\psi)$

Proof. We use in the proof trees a double line to denote omitted structural rules like weakening or exchange.

We prove $(T1)$ by induction on the structure of $\varphi(x)$. The other proofs are similar.

Base case: If $\varphi(x) = x$ then we have a cut-free G_0^*-proof for $\psi, \psi \longrightarrow \top$ by two consecutive application of weakening starting with the axiom $\longrightarrow \top$. We can immediately see that the size of the proof is $O(|\varphi(\psi)|)$.

Induction hypothesis: Suppose φ_i is an immediate subformula of φ. Then the sequent $\psi, \varphi_i(\psi) \longrightarrow \varphi_i(\top)$ has cut-free G_0^*-proof π_i of size $O(|\varphi_i(\psi)|^2)$.

Induction step: We distinguish the following cases according to the structure of φ.

Case 1. $\varphi(x) = \varphi_1(x) \wedge \varphi_2(x)$

By the induction hypothesis we have π_1 a proof of $\psi, \varphi_1(\psi) \longrightarrow \varphi_1(\top)$ and π_2 a proof of $\psi, \varphi_2(\psi) \longrightarrow \varphi_2(\top)$. A proof for $\psi, \varphi_1(\psi) \wedge \varphi_2(\psi) \longrightarrow \varphi_1(\top) \wedge \varphi_2(\top)$ can be constructed as follows.

$$
[\mathrm{El}]\;\dfrac{[\wedge\mathrm{r}]\;\dfrac{[\wedge\mathrm{l1}]\;\dfrac{[\mathrm{El}]\;\dfrac{\begin{matrix}\vdots\\ \pi_1\\ \vdots\\ \psi, \varphi_1(\psi) \longrightarrow \varphi_1(\top)\end{matrix}}{\varphi_1(\psi), \psi \longrightarrow \varphi_1(\top)}}{\varphi_1(\psi) \wedge \varphi_2(\psi), \psi \longrightarrow \varphi_1(\top)} \qquad \dfrac{\dfrac{\begin{matrix}\vdots\\ \pi_2\\ \vdots\\ \psi, \varphi_2(\psi) \longrightarrow \varphi_2(\top)\end{matrix}}{\varphi_2(\psi), \psi \longrightarrow \varphi_2(\top)}\;[\mathrm{El}]}{\varphi_1(\psi) \wedge \varphi_2(\psi), \psi \longrightarrow \varphi_2(\top)}\;[\wedge\mathrm{l2}]}{\varphi_1(\psi) \wedge \varphi_2(\psi), \psi \longrightarrow \varphi_1(\top) \wedge \varphi_2(\top)}}{\psi, \varphi_1(\psi) \wedge \varphi_2(\psi) \longrightarrow \varphi_1(\top) \wedge \varphi_2(\top)}
$$

Additionally to the π_i proofs (of size $O(|\varphi_i(\psi)|^2)$ the size is increased about $5 \cdot |\varphi_1(\psi)| + 5 \cdot |\varphi_2(\psi)| + 4 \cdot |\varphi_1(\top)| + 4 \cdot |\varphi_2(\top)| + 6 \cdot |\psi|$ (ignoring the constant number of additional logical symbols. For example in this proof there are 6 new occurrences of the $\wedge$ symbol).

This entails that we have a proof of size $O(|\varphi(\psi)|^2)$.

Case 2. Similarly one can prove the cases when $\varphi(x) = \varphi_1(x) \vee \varphi_2(x)$ and when $\varphi(x) = \neg\varphi_1(x)$.

Case 3. $\varphi(x) = \forall y \varphi_1(x, y)$

By the induction hypothesis we have π_1 a proof of $\psi, \varphi_1(\psi, z) \longrightarrow \varphi_1(\top, z)$. A proof of $\psi, \forall y \varphi_1(\psi, y) \longrightarrow \forall y \varphi_1(\top, y)$ is as follows.

$$[\forall\mathrm{r}]\,\dfrac{[\mathrm{El}]\,\dfrac{[\forall\mathrm{l}]\,\dfrac{[\mathrm{El}]\,\dfrac{\begin{matrix}\vdots\\ \pi_1\\ \vdots\\ \psi, \varphi_1(\psi, z) \longrightarrow \varphi_1(\top, z)\end{matrix}}{\varphi_1(\psi, z), \psi \longrightarrow \varphi_1(\top, z)}}{\forall y \varphi_1(\psi, y), \psi \longrightarrow \varphi_1(\top, z)}}{\psi, \forall y \varphi_1(\psi, y) \longrightarrow \varphi_1(\top, z)}}{\psi, \forall y \varphi_1(\psi, y) \longrightarrow \forall y \varphi_1(\top, y)}$$

The variable z is the eigenvariable of the $\forall$r inference.

Similar to the previous cases the size of the proof is still $O(|\varphi(\psi)|^2)$, as we had to increase the size of π_1 about $4 \cdot |\varphi_1(\psi, z)| + 4 \cdot |\varphi_1(\top, z)| + 4 \cdot |\psi|$ (ignoring the constant number of additional logical symbols).

Case 4. If $\varphi(x) = \exists y \varphi_1(x, y)$

By the induction hypothesis we have π_1 a proof of $\psi, \varphi_1(\psi, z) \longrightarrow \varphi_1(\top, z)$. A proof of $\psi, \exists y \varphi_1(\psi, y) \longrightarrow \exists y \varphi_1(\top, y)$ is as follows.

$$[\mathrm{El}]\,\dfrac{[\exists\mathrm{l}]\,\dfrac{[\mathrm{El}]\,\dfrac{[\exists\mathrm{r}]\,\dfrac{\begin{matrix}\vdots\\ \pi_1\\ \vdots\\ \psi, \varphi_1(\psi, z) \longrightarrow \varphi_1(\top, z)\end{matrix}}{\psi, \varphi_1(\psi, z) \longrightarrow \exists y \varphi_1(\top, y)}}{\varphi_1(\psi, z), \psi \longrightarrow \exists y \varphi_1(\top, y)}}{\exists y \varphi_1(\psi, y), \psi \longrightarrow \exists y \varphi_1(\top, y)}}{\psi, \exists y \varphi_1(\psi, y) \longrightarrow \exists y \varphi_1(\top, y)}$$

The variable z is the eigenvariable of the $\exists$l inference.

The size of the proof is again $O(|\varphi(\psi)|^2)$. □

Lemma 3.2.2. *G and* KPG *are p-equivalent. Moreover, for every $i \geq 1$,* KPG$_i$ *and* KPG$_i^*$ *are p-equivalent to G_i and G_i^*, respectively, for proving valid $\Sigma_i^q \cup \Pi_i^q$-formulas.*

Proof. The following proof is a detailed version of the proof in [CM05]. We prove that KPG$_i$ and G_i are p-equivalent for every $i \geq 0$. The proof is identical for KPG$_i^*$ versus G_i^* and KPG versus G.

KPG$_i$ p-simulates G_i

Restricting the target of ∀l and ∃r step in the systems G_i, G_i^* and G to be quantifier-free means that all quantifier introduction rules only increase quantifier complexity. This along with the fact that G_i is only allowed to cut formulas from $\Sigma_i^q \cup \Pi_i^q$ implies that a proof of a valid $\Sigma_i^q \cup \Pi_i^q$-formula in G_i will contain only $\Sigma_i^q \cup \Pi_i^q$-formula.

We can conclude that any G_i-proof of a $\Sigma_i^q \cup \Pi_i^q$-formula is in fact also a KPG$_i$-proof, which means that KPG$_i$ p-simulates G_i.

G_i p-simulates KPG$_i$

KPG$_i$-proofs contain $\Sigma_i^q \cup \Pi_i^q$-formulas only, while G_i-proofs could contain any QBF. However in G_i we are only allowed to cut $\Sigma_i^q \cup \Pi_i^q$-formulas.

This means that the allowed complexity of cut formulas in both systems is identical, thus we only need to prove that G_i can simulate the inference rules ∃r and ∀l with quantified targets.

Case ∃r. Note that we use here a double line to denote omitted structural rules like weakening or exchange.

Let S be the sequent $\Gamma \longrightarrow \Delta, \exists x\varphi(x)$ which is derived from $\Gamma \longrightarrow \Delta, \varphi(\psi)$ in KPG$_i$ such that ψ is quantified.

Using $(T1)$ and $(T2)$ from Lemma 3.2.1, we can get a proof of S in G_i as follows:

$$
\begin{prooftree}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$\Gamma \longrightarrow \Delta, \varphi(\psi)$}
\doubleLine
\RightLabel{[Wr]}
\UnaryInfC{$\Gamma \longrightarrow \Delta, \exists x\varphi(x), \varphi(\psi)$}
\AxiomC{proof of $(T2)$ $\vdots$}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\varphi(\psi) \longrightarrow \varphi(\bot), \psi$}
\doubleLine
\RightLabel{[∃r]}
\UnaryInfC{$\varphi(\psi) \longrightarrow \exists x\varphi(x), \psi$}
\AxiomC{proof of $(T1)$ $\vdots$}
\noLine
\UnaryInfC{$\vdots$}
\noLine
\UnaryInfC{$\psi, \varphi(\psi) \longrightarrow \varphi(\top)$}
\LeftLabel{[∃r]}
\UnaryInfC{$\psi, \varphi(\psi) \longrightarrow \exists x\varphi(x)$}
\LeftLabel{[Cut ψ]}
\BinaryInfC{$\varphi(\psi) \longrightarrow \exists x\varphi(x)$}
\doubleLine
\LeftLabel{[W]}
\UnaryInfC{$\varphi(\psi), \Gamma \longrightarrow \Delta, \exists x\varphi(x)$}
\RightLabel{[Cut $\varphi(\psi)$]}
\BinaryInfC{$\Gamma \longrightarrow \Delta, \exists x\varphi(x)$}
\end{prooftree}
$$

Note that we are allowed to cut on $\varphi(\psi)$ and ψ because we know that both are in $\Sigma_i^q \cup \Pi_i^q$, by the definition of KPG$_i$.

Moreover, the conditions for applying the rule ∃r are met, since the rules are applied to quantifier-free target formulas, namely $\top, \bot$.

We finally note that this proof is of size polynomial in the size of the KPG$_i$-proof, since the proofs of T_1 and T_2 are each of size $O(|\varphi(\psi)|^2)$ and we increased them about $(9+\alpha)\cdot|\varphi(x)| + (8+\alpha)\cdot|\varphi(\psi)| + 4\cdot|\psi| + (4+\alpha)\cdot|\Delta| + (4+\alpha)\cdot|\Gamma|$, such that α is the number of formulas in Δ and Γ (which we add in the right side of the proof by weakening).

Case ∀l. Assume that we get the sequent $\forall x\varphi(x), \Gamma \longrightarrow \Delta$ from $\varphi(\psi), \Gamma \longrightarrow \Delta$ where ψ is quantified. Then the proof of S in G_i is similar to the previous case with the modification that we start with $(T3)$ and $(T4)$ from Lemma 3.2.1. □

Definition 3.2.9 (Free-variable Normal Form)
A treelike proof π is in *free-variable normal form* if

- no parameter variable is used as an eigenvariable in π, and
- every non-parameter variable is used as an eigenvariable exactly once in π. ◁

If a proof π is treelike and in free-variable normal form, then for every non-parameter variable b, the sequents containing b form a sub-tree of π, whose end-sequent is the upper sequent of the inference in which b is used as an eigenvariable.

Buss in [Bus98] and other works mentions that we can convert any treelike proof into a proof in free-variable normal form by simply renaming variables.

In what follows we assume that every treelike proof is in free variable normal form.

CHAPTER 4

Bounded Arithmetic

This chapter introduces a family of bounded arithmetic theories which are fragments of Peano arithmetic. We define a translation of these formulas to QBFs.

To establish the connection between bounded arithmetic and the polynomial hierarchy, S. Buss defines, in chapter 1 of [Bus86], the polynomial hierarchy within the setting of bounded quantification, and the class of polynomially bounded functions which are closed under composition and are defined using the concept of logarithmically (and polynomially) bounded quantification.

Buss then introduces two types of bounded quantifiers which corresponds to the logarithmically (and polynomially) bounded quantification, namely the bounded quantifiers of the form $\forall x \leq t\ A(x)$ or $\exists x \leq t\ A(x)$ (which mean $\forall x(x \leq t \supset A(x))$ and $\exists x(x \leq t \wedge A(x))$, respectively), and sharply bounded quantifiers (a special case of the former) of the form $\forall x \leq |t|\ A(x)$ and $\exists x \leq |t|\ A(x)$. He then defines a hierarchy of bounded arithmetic formulas based on the number of bounded quantifier alternations while ignoring sharply bounded ones, which we will present in this chapter.

An important feature of bounded arithmetic is that computing the truth value of a bounded arithmetic formula is always bounded by some polynomial $q(\cdot)$ with respect to the size of the input, and that is because the terms are constructed from polynomial time computable functions and predicates as seen in the later parts of the thesis.

Before we dive in the first-order theories of bounded arithmetic, we recall the definition of a well-formed first-order formula.

4.1 Syntax of first-order logic

The syntax of first-order logic is defined over a set of symbols, these symbols can be divided to

- logical symbols, which are $\{\neg, \vee, \wedge, \supset, \exists, \forall\}$, and
- non-logical symbols, which include constant, function, and predicate symbols.

Additionally the symbols $x, y, z, x_1, x', \ldots$ are used to denote variables.

Definition 4.1.1
A *term* for a given set of symbols can be defined inductively as follows

- A constant, or a variable is a term.
- If $t_1, \ldots, t_n$ are terms, then, for any function symbol f of arity n, $f(t_1, \ldots, t_n)$ is a term. $\triangleleft$

Definition 4.1.2
A first-order *formula* for a given set of symbols can be defined inductively as follows

- If $t_1, \ldots, t_n$ are terms, then, for any predicate symbol P of arity n, $P(t_1, \ldots, t_n)$ is a formula.
- If $\varphi, \varphi_1, \varphi_2$ are formulas, then $\neg\varphi$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \supset \varphi_2$, $\exists x \varphi$, and $\forall x \varphi$ are also formulas. $\triangleleft$

4.2 The language of bounded arithmetic

The first-order language of bounded arithmetic, denoted by L, contains the following symbols.

Logical Symbols		
$\wedge$	And	$A \wedge B$
$\vee$	Or	$A \vee B$
$\neg$	Not	$\neg A$
$\supset$	Implication	$A \supset B$
$\exists$	Exists	$\exists x A(x)$
$\forall$	For all	$\forall x A(x)$
Function Symbols		
0	Zero constant	
S	Successor function	$S(x) = x + 1$
$+$	Addition function	$x + y$

$\cdot$	Multiplication function	$x \cdot y$
$\lvert\ \rvert$	The length of the binary representation	$\lvert x\rvert = \lceil log_2(x+1)\rceil$
$\lfloor \frac{1}{2}\ \rfloor$	Divide by two and round down	$\lfloor \frac{1}{2}x\rfloor$
#	Smash function	$x\#y = 2^{\lvert x\rvert\cdot\lvert y\rvert}$
Predicate Symbols		
$=$	Equality	
$\leq$	Less or equal	
Other Symbols		
( )	Parentheses	

The function symbols take non-negative integers as input, x and y are variables, and A, B are formulas.

Definition 4.2.1 (Bounded quantifiers, bounded formula)
Quantifiers of the form $\exists x$ and $\forall x$ are called unbounded quantifiers. We define two types of bounded quantifiers.

i. A *bounded quantifier* is of the form $\exists x \leq t$ or $\forall x \leq t$, in which t can be any term not involving x.

ii. A *sharply bounded quantifier* is of the form $\exists x \leq |t|$ or $\forall x \leq |t|$, such that t is again any term not involving x.

A *bounded formula* is a formula with no unbounded quantifiers.

A *sharply bounded formula* is a formula containing sharply bounded quantifiers only. $\triangleleft$

We define a hierarchy of bounded formulas as follows:

Definition 4.2.2 (Definition from Buss in [Bus86])
Σ^b_{k+1} and Π^b_{k+1} are the smallest sets which satisfy the following conditions.

(1.) $\Pi^b_0 = \Sigma^b_0 = \Delta^b_0$ is the set of sharply bounded formulas of L.

(2.) Σ^b_{k+1} is defined as follows :

(a.) $\Pi^b_k \subseteq \Sigma^b_{k+1}$.
(b.) If A is in Σ^b_{k+1} then so are $(\exists x \leq t\ A(x))$ and $(\forall x \leq |t|\ A(x))$ for any term t not involving x, where x is free in A.
(c.) If $A, B \in \Sigma^b_{k+1}$ then $A \wedge B$ and $A \vee B$ are in Σ^b_{k+1}.
(d.) If $A \in \Sigma^b_{k+1}$ and $B \in \Pi^b_{k+1}$ then $\neg B$ and $B \supset A$ are in Σ^b_{k+1}.

(3.) Π^b_{k+1} is defined as follows:

(a.) $\Sigma^b_k \subseteq \Pi^b_{k+1}$.
(b.) If A is in Π^b_{k+1} then so are $(\forall x \leq t\ A(x))$ and $(\exists x \leq |t|\ A(x))$ for any term t not involving x, where x is free in A.

(c.) If $A, B \in \Pi^b_{k+1}$ then $A \wedge B$ and $A \vee B$ are in Π^b_{k+1}.
(d.) If $A \in \Pi^b_{k+1}$ and $B \in \Sigma^b_{k+1}$ then $\neg B$ and $B \supset A$ are in Π^b_{k+1}. ◁

Note that the classes of bounded formulas in this hierarchy are based on the alternations of bounded quantifiers only, thus the alternations of sharply bounded quantifiers do not count.

4.3 Axiomatization of bounded arithmetic

Peano arithmetic is usually axiomatized by a small number of axiom schemata and an induction schema. Buss in [Bus86] axiomatized bounded arithmetic by increasing the number of axioms and restricting the induction axioms.

We first define the theory called BASIC.
Definition 4.3.1 (The theory BASIC)
BASIC is a finite set of axioms, which define the simple properties relating the function and predicate symbols of the language L (the language of bounded arithmetic).

BASIC consists of the following 32 axioms:

1. $|0| = 0$
2. $0 \leq x$
3. $x + 0 = x$
4. $x \cdot 0 = 0$
5. $|S(0)| = S(0)$
6. $x \neq S(x)$
7. $x + y = y + x$
8. $x \leq x + y$
9. $y \leq x \supset y \leq S(x)$
10. $x \neq 0 \supset 2 \cdot x \neq 0$
11. $x \leq y \vee y \leq x$
12. $x \leq y \supset |x| \leq |y|$
13 .$(x \leq y \wedge y \leq x) \supset x = y$
14. $(x \leq y \wedge y \leq z) \supset x \leq z$
15. $(x \leq y \wedge x \neq y) \supset S(x) \leq y$
16. $(x \leq y \wedge x \neq y) \supset (S(2 \cdot x) \leq 2 \cdot y \wedge S(2 \cdot x) \neq 2 \cdot y)$
17. $x \neq 0 \supset (1\#(2 \cdot x) = 2 \cdot (1\#x) \wedge 1\#(S(2 \cdot x)) = 2 \cdot (1\#x))$
18. $x + y \leq x + z \supset y \leq z$
19. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
20. $x \neq 0 \supset |x| = S(|\lfloor \frac{x}{2} \rfloor|)$
21. $x \cdot (S(y)) = (x \cdot y) + x$
22. $x \cdot y = y \cdot x$
23. $(x + y) + z = x + (y + z)$
24. $x + S(y) = S(x + y)$
25. $0\#x = S(0)$
26. $|x| = |y| \supset x\#z = y\#z$
27. $|x| = |y| + |z| \supset x\#u = (y\#u) \cdot (z\#u)$
28. $x\#y = y\#x$
29. $|x\#y| = S(|x| \cdot |y|)$
30. $S(0) \leq x \supset (x \cdot y \leq x \cdot z \longleftrightarrow y \leq z)$
31. $x = \lfloor \frac{y}{2} \rfloor \longleftrightarrow (2 \cdot x = y \vee S(2 \cdot x) = y)$
32. $x \neq 0 \supset (|2 \cdot x| = S(|x|) \wedge |S(2 \cdot x)| = S(|x|))$

◁

Definition 4.3.2 (Induction axioms)

Let Ψ be any set of formulas of L. We define three types of induction axioms.

The general induction axiom Ψ-IND:

$$\text{For any } A \in \Psi : A(0) \wedge \forall x\ (A(x) \supset A(S(x))) \supset \forall x\ A(x).$$

The polynomial induction axiom Ψ-PIND:

$$\text{For any } A \in \Psi : A(0) \wedge \forall x\ (A(\left\lfloor \frac{x}{2} \right\rfloor) \supset A(x)) \supset \forall x\ A(x).$$

The length induction axiom Ψ-LIND:

$$\text{For any } A \in \Psi : A(0) \wedge \forall x\ (A(x) \supset A(S(x))) \supset \forall x\ A(|x|). \qquad \triangleleft$$

We note that while we will use the axioms Ψ-IND and Ψ-PIND in the definition of theories S_2 and T_2, we will not need the axiom Ψ-LIND in this thesis. However we mention it to complete the picture.

In what follows we define, over the language L, fragments of bounded arithmetic known as the theories T_2 and S_2.

4.3.1 Theories S_2 and T_2 of bounded arithmetic

Definition 4.3.3 (The theory T_2)
T_2^i is the theory BASIC extended by Σ_i^b-IND (i.e, the induction axiom for all Σ_i^b-formulas).

The theory T_2 is the union of all theories T_2^i. $\triangleleft$

Definition 4.3.4 (The theory S_2)
S_2^i is the theory BASIC extended by Σ_i^b-PIND (i.e, the polynomial induction axiom for all Σ_i^b-formulas).

The theory S_2 is the union of all theories S_2^i. $\triangleleft$

We note that the theory BASIC is sometimes denoted by $T_2^{(-1)}$ or $S_2^{(-1)}$ in the literature.

4.4 Sequent-like calculi for bounded arithmetic

Takeuti in [Tak87] defines the calculus LK$_e$ for predicate logic with equality by extending the propositional PK with quantifier rules and adding the following sequents as additional possible initial sequents.

For any term s

$$\longrightarrow s = s,$$

for every function symbol f

$$t_1 = s_1, \dots, t_k = s_k \longrightarrow f(t_1, \dots, t_k) = f(s_1, \dots, s_k),$$

and or every predicate symbol R

$$t_1 = s_1, \dots, t_k = s_k, R(t_1, \dots, t_k) \longrightarrow R(s_1, \dots, s_k),$$

such that $t_1, \dots, t_k$ and $s_1, \dots, s_k$ could be any term.

The rules introducing first-order quantifiers are as follows.

$$[\exists\mathrm{l}]\ \frac{A(a), \Gamma \longrightarrow \Delta}{\exists x A(x), \Gamma \longrightarrow \Delta} \qquad [\exists\mathrm{r}]\ \frac{\Gamma \longrightarrow \Delta, A(t)}{\Gamma \longrightarrow \Delta, \exists x A(x)}$$

$$[\forall\mathrm{l}]\ \frac{A(t), \Gamma \longrightarrow \Delta}{\forall x A(x), \Gamma \longrightarrow \Delta} \qquad [\forall\mathrm{r}]\ \frac{\Gamma \longrightarrow \Delta, A(a)}{\Gamma \longrightarrow \Delta, \forall x A(x)}$$

The term t is any term free for x in A, and a is an eigenvariable that does not occur free in the lower sequent. Furthermore we assume for simplicity that the formula A in the axiom $A \longrightarrow A$ is atomic.

Lemma 4.4.1 (Takeuti §7. in [Tak87])**.**

Let $A(a_1, \dots, a_k)$ be an arbitrary formula, then the sequents

$$t_1 = s_1, \dots, t_k = s_k, A(t_1, \dots, t_k) \longrightarrow A(s_1, \dots, s_k),$$

$$s = t \longrightarrow t = s, \ \textit{and}$$

$$s_1 = s_2, s_2 = s_3 \longrightarrow s_1 = s_3$$

are provable in LK_e *such that* $s, t, t_1, \dots, t_k$ *and* $s_1, \dots, s_k$ *are terms.*

Now we define the system LKB for bounded arithmetic formulas by extending LK_e with rules allowing the introduction of bounded quantifiers.

$$[\exists \le\mathrm{l}]\ \frac{a \le t,\ A(a), \Gamma \longrightarrow \Delta}{\exists x \le t\ A(x), \Gamma \longrightarrow \Delta} \qquad [\exists \le\mathrm{r}]\ \frac{\Gamma \longrightarrow \Delta, A(t)}{t \le s,\ \Gamma \longrightarrow \Delta, \exists x \le s\ A(x)}$$

$$[\forall \le\mathrm{l}]\ \frac{A(t), \Gamma \longrightarrow \Delta}{t \le s,\ \forall x \le s\ A(x), \Gamma \longrightarrow \Delta} \qquad [\forall \le\mathrm{r}]\ \frac{a \le t,\ \Gamma \longrightarrow \Delta, A(a)}{\Gamma \longrightarrow \Delta, \forall x \le t\ A(x)}$$

The terms t and s can be any terms not containing x, a does not occur in t, and the variable a must not occur free in the lower sequent.

We can also introduce inference rules representing the induction axioms.

Definition 4.4.1
We define the induction inference rules as follows.

(a) IND-rule

$$\frac{A(a), \Gamma \longrightarrow \Delta, A(a+1)}{A(0), \Gamma \longrightarrow \Delta, A(t)}$$

(b) PIND-rule

$$\frac{A(\lfloor \frac{x}{2} \rfloor), \Gamma \longrightarrow \Delta, A(a)}{A(0), \Gamma \longrightarrow \Delta, A(t)}$$

(c) LIND-rule

$$\frac{A(a), \Gamma \longrightarrow \Delta, A(a+1)}{A(0), \Gamma \longrightarrow \Delta, A(|t|)}$$

The term t is any term and the variable a must not occur free in the lower sequent.

The concept of auxiliary formulas, principal formulas, and successors is the same as introduced in the definition of PK in Chapter 3. Note however that each of the induction rules has two *auxiliary formulas*, e.g., $A(a)$ and $A(a+1)$ in the IND-rule, and two *principal formulas*, e.g., $A(0)$ and $A(t)$ in IND-rule. $\triangleleft$

According to J. Krajíček, in Chapter 7 of [Kra95], the system LKB is sound and complete, and any valid sequent formed from bounded formulas can be proven in LKB without using unbounded quantifiers.

By restricting the formulas in the above rules to those from Σ_i^b, we get rules equivalent to the Σ_i^b-IND (resp. Σ_i^b-PIND, Σ_i^b-LIND) axioms introduced in Definition 4.3.2, a fact that was proven by Buss in [Bus88] by deriving the sequent $\longrightarrow A(0) \wedge \forall x\ (A(x) \supset A(S(x))) \supset \forall x\ A(x)$ in LKB with the IND-rule restricted to Σ_i^b-formulas (similarly for PIND and LIND).

An important property of sequent calculi is the possibility to eliminate applications of the cut rule, which was proven by Gentzen for LK (PK + the quantifier rules, but without equality) in his famous Hauptsatz [Gen35]. It states that each LK-derivation can be transformed to a cut-free LK-derivation with the same end-sequent.

However eliminating the cut rule completely is not possible for LKB, but its application can be restricted to formulas satisfying specific properties as was proven by Takeuti in [Tak87]. For example in LK$_e$ we can eliminate all cuts except those in which the cut formula is of the form $t = s$.

Definition 4.4.2
A formula A in an LKB derivation with induction rules is called *free*, if A is an axiom or if no successor of A is identical to A or to one of the two principal formulas of an induction rule. ◁

Theorem 4.4.2 ([Kra95]). *Assume $i \geq 1$.*

If the sequent $\Gamma \longrightarrow \Delta$ is provable in T_2^i then it has an LKB*-proof with the* Σ_i^b-IND *rule in which no cut formula is free.*

The same holds for S_2^i and the Σ_i^b-PIND *rule.*

A corollary of the original cut elimination is the subformula property which states that every formula occurring in a derivation is a subformula of the end-sequent. However, just like the cut elimination theorem, only a weaker form of it is applicable in LKB, which is the following.

Corollary 4.4.2.1 ([Kra95]). *For every $i \geq 1$, if $\Gamma \longrightarrow \Delta$ is provable in T_2^i (resp. in S_2^i), (i.e, if all formulas in Γ, Δ are Σ_i^b-formulas), then all formulas in the proof are Σ_i^b- or Π_i^b-formulas.*

Before we present the translation of bounded arithmetic formulas over L to QBFs, we rely on [Bus88] in what follows to define the bounding polynomial of a formula A of S_2^1.

Definition 4.4.3
The bounding polynomial $q_t(n)$ of a term t over the language L, is defined inductively as follows.

- $q_0(n) = 1$
- $q_x(n) = n$ for any variable x
- $q_{S(t)}(n) = q_t(n) + 1$, where S is the successor function
- $q_{t+u}(n) = q_t(n) + q_u(n)$
- $q_{t\cdot u}(n) = q_t(n) + q_u(n)$
- $q_{t\#u}(n) = q_t(n) \cdot q_u(n) + 1$
- $q_{|t|}(n) = q_{\lfloor \frac{t}{2} \rfloor}(n) = q_t(n)$

The bounding polynomial q_A of a bounded arithmetic formula A over L, is defined inductively as follows.

- $q_{t=u} = q_{t\leq u} = q_t + q_u$

- $q_{A \wedge B} = q_{A \vee B} = q_{A \supset B} = q_A + q_B$
- $q_{\neg A} = q_A$
- $q_{\exists x \leq t A}(n) = q_{\forall x \leq t A}(n) = q_t(n) + q_A(n + q_t(n))$

Note that by definition $|t(x_1, \ldots, x_k)| \leq q_t(n)$ and $A(x_1, \ldots, x_k)$ refers only to numbers of length $\leq q_A(n)$ where $|x_i| \leq n$ for $i \in \{1, \ldots, k\}$. $\triangleleft$

4.5 The Translation of Bounded Arithmetic to QBFs

The language of bounded arithmetic L is polynomial-time recognizable, thus by Theorem 2.2.1 any term $t(a_1, \ldots, a_k)$ from L, for $|a_1|, \ldots, |a_k| \leq m$, can be computed by a Boolean circuit C_t in de Morgan basis *i.e.*, $\{0, 1, \neg, \wedge, \vee\}$, such that C_t is of size polynomial in m.

This means that the circuit C_t computing the term t of the language L can be translated to a Σ_1^q-QBF after introducing new atoms for each node in the circuit C_t. We use the symbols $p, q, r, u, \ldots$ to denote the introduced atoms, and $\varphi_t^m(\overline{p}_1, \ldots, \overline{p}_k, \overline{q})$ to denote the Σ_1^q-QBF, which expresses the statement "There is a computation of the circuit C_t, which outputs $\overline{q}$ on inputs $\overline{p}_1, \ldots, \overline{p}_k$".

In what follows we use the following notations.

- $a(i) \in \{0, 1\}$ is the i-th digit of the binary representation of the integer a (i.e, $a = \sum\limits_{i \leq |a|} a(i) \cdot 2^i$), such that if $i > |a|$ then $a(i) = 0$.
- $\varphi(n)$ and $\varphi(\overline{p}/n)$ both denote the formula $\varphi(p_0/n(0), p_1/n(1), \ldots)$, such that φ is a QBF with the free atoms $\overline{p} = (p_0, p_1, \ldots)$.

Definition 4.5.1 (The Translation by Krajíček, Chapter 9 in [Kra95])
Let $A(a_1, \ldots, a_k)$ be a bounded formula in the language L of S_2, where $|a_1|, \ldots, |a_k| \leq m$. Let $q(\cdot)$ be a bounding polynomial for the formula A. Moreover let

$$X := \{\overline{\epsilon} \in \{0, 1\}^* \mid \epsilon_i = 0 \text{ for } i > |q(m)|\}.$$

For every m we construct a quantified Boolean formula $[[A]]_{q(m)}^m$ with the atoms $\overline{p}_i$ such that for every $i \in \{1, \ldots, k\}$ we have $\overline{p}_i = (p_i^0, \ldots, p_i^{q(m)})$. In what follows we define $[[A]]_{q(m)}^m$ inductively.

1. For A the atomic formula $t(\overline{a}) = s(\overline{a})$:

$$[[A]]_{q(m)}^m := \exists x_0, \ldots, x_{q(m)}, y_0, \ldots, y_{q(m)}\ \varphi_t(\overline{p}_1, \ldots, \overline{p}_k, q_j/x_j) \ \wedge \ \varphi_s(\overline{p}_1, \ldots, \overline{p}_k, q_j/y_j) \ \wedge \bigwedge_{0 \leq i \leq q(m)} x_i \leftrightarrow y_i$$

   where $x \leftrightarrow y$ has the same meaning as $(x \supset y) \wedge (y \supset x)$.

2. For A the atomic formula $t(\overline{a}) \leq s(\overline{a})$:
$$[[A]]^m_{q(m)} := \exists x_0, \ldots, x_{q(m)}, y_0, \ldots, y_{q(m)}\ \varphi_t(\overline{p}_1, \ldots, \overline{p}_k, q_j/x_j)\ \wedge$$
$$\varphi_s(\overline{p}_1, \ldots, \overline{p}_k, q_j/y_j)\ \wedge \bigwedge_{0 \leq i \leq q(m)} (((\bigwedge_{i+1 \leq j \leq q(m)} x_j \leftrightarrow y_j) \wedge x_i) \supset y_i).$$
Note that the last conjunct defines the lexicographic order on $\overline{x}, \overline{y}$.

3. For $A = \neg A_1$ then
$$[[A]]^m_{q(m)} := \neg [[A_1]]^m_{q(m)}.$$

4. For $A = A_1 \circ A_2$ where $\circ \in \{\wedge, \vee\}$ then
$$[[A]]^m_{q(m)} := [[A_1]]^m_{q(m)} \circ [[A_2]]^m_{q(m)}.$$

5. For $A(a) = \exists x \leq |t|\ A_1(a, x)$ then
$$[[A]]^m_{q(m)} := \bigvee_{\overline{\epsilon} \in X} [[b \leq |t| \wedge A_1(a, b)]]^m_{q(m)}(\overline{q}/\overline{\epsilon}).$$
For $A(a) = \forall x \leq |t|\ A_1(a, x)$ then
$$[[A]]^m_{q(m)} := \bigwedge_{\overline{\epsilon} \in X} [[b \leq |t| \supset A_1(a, b)]]^m_{q(m)}(\overline{q}/\overline{\epsilon})$$
where $\overline{q}$ is the tuple associated with b.

6. For $A(a) = \exists x \leq t\ A_1(a, x)$, in which t is not of the form $|s|$ then
$$[[A]]^m_{q(m)} := \exists x_0, \ldots, x_{q(m)} [[b \leq t \wedge A_1(a, b)]]^m_{q(m)}(\overline{q}/\overline{x}).$$
For $A(a) = \forall x \leq t\ A_1(a, x)$, in which t is not of the form $|s|$ then
$$[[A]]^m_{q(m)} := \forall x_0, \ldots, x_{q(m)} [[b \leq t \supset A_1(a, b)]]^m_{q(m)}(\overline{q}/\overline{x})$$
where $\overline{q}$ is the tuple associated with b. $\triangleleft$

Many lemmas, corollaries, and theorems presented in what follows, which describe the strong connection between KPG and bounded arithmetic, were introduced by Krajícek and Pudlák in [Kra95] and [KP90]. However, even though those results were proven for KPG, they are applicable for G as well. We already proved in Lemma 3.2.2 that KPG and G are p-equivalent, and for any $i \geq 0$, KPG$_i$ and G_i are p-equivalent in proving $\Sigma^q_i \cup \Pi^q_i$-formulas.

The accurate notation for the translation of a formula $A(a)$ is $[[A(a)]]^m_{q(m)}(\overline{p})$, where $\overline{p}$ are the atoms associated with the variable a. However, in what follows we omit some elements of the notation for simplicity (e.g., $m, q(m)$, and $(\overline{p})$). As a reminder, and when necessary, we add the atoms associated with the variables, specifically when those atoms are substituted by some $\overline{\epsilon} \in X$.

Lemma 4.5.1 (Lemma 9.2.3 in [Kra95]). *Let $A \in \Sigma^b_0$, t be a term, a a free variable, and $q(\cdot)$ a bounding polynomial for $A(t)$. Then for every m, there are size $m^{O(1)}$ KPG$^*_1$-proofs of*
$$[[t = a \supset A(a)]]^m_{q(m)} \longrightarrow [[A(t)]]^m_{q(m)}.$$

Though we do not prove this lemma here, we note that the system KPG$_0$ is not powerful enough to derive this sequent, mainly because the sequent contains a Σ_1^q-formula, which is not allowed in KPG$_0$. Recall that by definition of the translation $[[t = a]]$ is a Σ_1^q-formula.

Krajíček, in the proof of Theorem 9.2.5 in [Kra95], uses a more general version of this lemma, and that is when $A \in \Sigma_i^b$ (in that case the proof of the sequent would be at least a KPG$_i^*$-proof). A restricted version of this lemma (with a conjunction on the left side instead of implication) is presented in [KP90] within the definition of the translation. In what follows we present it as corollary, and we prove it starting from the previous lemma. We also prove it for the general case when $A \in \Sigma_i^b$ for $i \geq 0$, but in the system G_1^* (not KPG$_1^*$).

Corollary 4.5.1.1. *Let $A \in \Sigma_0^b$, t be a term, a a free variable, and $q(\cdot)$ a bounding polynomial for $A(t)$. Then for every m, there are size $m^{O(1)}$ KPG$_1^*$-proofs of*

$$[[t = a \land A(a)]]^m_{q(m)} \longrightarrow [[A(t)]]^m_{q(m)}.$$

Proof.

We start the proof as follows.

$$[\supset l]\ \dfrac{[\supset r]\ \dfrac{\dfrac{[[A(a)]] \longrightarrow [[A(a)]]}{[[t=a]],[[t=a]],[[A(a)]] \longrightarrow [[A(t)]],[[A(a)]]}}{[[t=a]],[[A(a)]] \longrightarrow [[A(t)]],[[t = a \supset A(a)]]} \qquad \dfrac{[[A(t)]] \longrightarrow [[A(t)]]}{[[A(t)]],[[t=a]],[[A(a)]] \longrightarrow [[A(t)]]}}{[[(t = a \supset A(a)) \supset A(t)]],[[t=a]],[[A(a)]] \longrightarrow [[A(t)]]} \quad \cdots\ S_0$$

By Lemma 4.5.1 and the rule $[\supset r]$ we derive $\longrightarrow [[(t = a \supset A(a)) \supset A(t)]] \qquad \cdots\ S_1$

By applying cut on S_0 and S_1 we get $[[t = a]], [[A(a)]] \longrightarrow [[A(t)]]$, then by applying $\land$l1, $\land$l2, and contraction on this sequent we obtain

$$[[t = a \land A(a)]] \longrightarrow [[A(t)]]. \qquad \square$$

In the following lemma we prove that a G_1^*-proof of the above sequent exists when $A \in \Sigma_i^q$ for any $i \geq 0$. This is possible in G_1^* because we do not use cut on any formula in Σ_i^q when $i > 1$, though the proof contains such formulas, which is why such a proof can not be a KPG$_1^*$-proof. However, with minor changes in the proof below (like the induction hypothesis, for example) we can prove that a KPG$_i^*$-proof of the sequent exits.

Lemma 4.5.2. *Let $A \in \Sigma_i^b$ for any $i \geq 0$, t be a term, a a free variable, and $q(\cdot)$ a bounding polynomial for $A(t)$. Then for every m there are size $m^{O(1)}$ G_1^*-proofs of*

$$[[t = a \land A(a)]]^m_{q(m)} \longrightarrow [[A(t)]]^m_{q(m)}.$$

Proof.

For any $A \in \Sigma_i^q$, such that $i \geq 0$, we can get the sequent $[[t = a \land A(a)]] \longrightarrow [[A(t)]]$, by applying $\land$l1, $\land$l2, and contraction on the sequent

$$[[t = a]], [[A(a)]] \longrightarrow [[A(t)]] \quad \cdots \; S$$

In what follows we prove by mathematical induction on i, that the sequent S has a G_1^*-proof of size $m^{O(1)}$, for each $i \geq 0$.

Base Case

For the case when $i = 0$, the proof is same of corollary 4.5.1.1 before applying the conjunction rules.

Induction hypothesis

Let $i \geq 0$ and assume that for $A \in \Sigma_i^b$, the sequent $[[t = a]], [[A(a)]] \longrightarrow [[A(t)]]$ has a G_1^*-proof of size $m^{O(1)}$.

Induction Step

By the definition of the hierarchy of bounded formulas, if $B_1, B_2 \in \Sigma_i^b$ then so are $B_1 \land B_2, B_1 \lor B_2, \exists x \leq t \; B_1(X)$, and $\forall x \leq |t| \; B_1(x)$. Moreover, if $C \in \Pi_i^b$ then $\neg C$ and $C \supset B_1$ are also in Σ_i^b.

Thus to prove the lemma for $A \in \Sigma_{i+1}^b$, we need to prove it when A is one of the following two cases.

Case 1 $A = \forall x \leq s \; B(a, x)$

By the induction hypothesis $[[t = a]], [[B(a, b)]] \longrightarrow [[B(t, b)]]$ has a G_1^*-proof of size $m^{O(1)}$, for some variable b, such that neither a nor b occurs in t.

Let $\overline{q}$ be the tuple associated with b (as in the definition of the translation above). We then get the required proof as follows.

We first get the sequent S_1:

$$[\supset\text{r}] \; \begin{array}{c} [[b \leq s]] \longrightarrow [[b \leq s]] \\ \hline\hline [[b \leq s]], [[t = a]] \longrightarrow [[b \leq s]], [[B(t, b)]] \\ \hline [[t = a]] \longrightarrow [[b \leq s \supset B(t, b)]], [[b \leq s]] \end{array} \quad \cdots \; S_1$$

We then derive the sequent S_2:

$$\begin{array}{c} [[t = a]], [[B(a, b)]] \longrightarrow [[B(t, b)]] \\ \hline [[b \leq s]], [[t = a]], [[B(a, b)]] \longrightarrow [[B(t, b)]] \\ \hline [[t = a]], [[B(a, b)]] \longrightarrow [[b \leq s \supset B(t, b)]] \end{array} \begin{array}{l} [\text{Wl}] \\ \\ [\supset\text{r}] \end{array} \quad \cdots \; S_2$$

By applying $\supset$l with the appropriate structural rules on S_1 and S_2, we get

$$[[t = a]], [[b \leq s \supset B(a,b)]] \longrightarrow [[b \leq s \supset B(t,b)]] \qquad \cdots\cdot \; S_3$$

Starting from S_3, we then can derive the needed sequent by applying $\forall$l and $\forall$r as follows.

$$\dfrac{\dfrac{[[t = a]], [[b \leq s \supset B(a,b)]] \longrightarrow [[b \leq s \supset B(t,b)]]}{[[t = a]], \forall\overline{x}\ [[b \leq s \supset B(a,b)]](\overline{q}/\overline{x}) \longrightarrow [[b \leq s \supset B(t,b)]]}\,[\forall \mathrm{l}]}{[[t = a]], \forall\overline{x}\ [[b \leq s \supset B(a,b)]](\overline{q}/\overline{x}) \longrightarrow \forall\overline{x}[[b \leq s \supset B(t,b)]](\overline{q}/\overline{x})}\,[\forall \mathrm{r}]$$

We recall that by the definition of the translation the end-sequent is the same as

$$[[t = a]], [[\forall x \leq s\ B(a,x)]] \longrightarrow [[\forall x \leq s\ B(t,x)]].$$

Case 2 $A = \exists x \leq |s|\ B(a,x)$
In a similar way like in the previous case, by using weakening, exchange, and $\supset$ left and right, we get for each $\overline{\epsilon} \in X$:

$$[[t = a]], [[b \leq |s| \supset B(a,b)]](\overline{q}/\overline{\epsilon}) \longrightarrow [[b \leq |s| \supset B(t,b)]](\overline{q}/\overline{\epsilon}).$$

Then by repeatedly applying $\vee$r and then $\vee$l, along with the appropriate structural rules we get a proof of

$$[[t = a]], \bigvee_{\overline{\epsilon} \in X} [[b \leq |s| \supset B(a,b)]](\overline{q}/\overline{\epsilon}) \longrightarrow \bigvee_{\overline{\epsilon} \in X} [[b \leq |s| \supset B(t,b)]](\overline{q}/\overline{\epsilon}).$$

By the definition of the translation we see that the end-sequent is the same as

$$[[t = a]], [[\exists x \leq |s|\ B(a,x)]] \longrightarrow [[\exists x \leq |s|\ B(t,x)]]. \qquad \square$$

In what follows we introduce Lemmas 4.5.4 and 4.5.5, in addition to Lemma 4.5.3 to be used in the proof of the main theorem in this chapter, Theorem 4.5.6. In this theorem we see how an LKB-proof of a formula $A \in \Sigma_i^b$ can be reconstructed to get a KPG$_i$-proof (thus a G_i-proof) of the translation of A. However, in the proofs, we occasionally use what is known as the *substitution rule*, which allows simultaneous substitution of formulas for atoms in one inference step. This rule is usually used as an extension to Frege systems. Krajícek and Pudlák in Lemma 2.1. in [KP90] use a special case of the substitution rule and prove that it can be polynomially simulated in KPG and KPG$_i$ for any $i \geq 1$. The substitution rule they use is

$$[\mathrm{sub}]\ \dfrac{\Gamma(a) \longrightarrow \Delta(a)}{\Gamma(\varphi) \longrightarrow \Delta(\varphi)}$$

such that φ is quantifier-free, a does not occur in φ, and all occurrences of a in Γ and Δ are substituted with φ.

Note that we do not always mention the use of structural rules, specially the use of contraction and exchange. It is important to keep that in mind since these rules are often applied in the following proofs, but not mentioned.

Lemma 4.5.3 (Lemma 9.2.4 in [Kra95])**.** *Let A be an axiom of* BASIC *and $q(\cdot)$ a bounding polynomial of A, then, for all m, there are size $m^{O(1)}$* KPG$_1^*$*-proofs of the formula $[[A]]^m_{q(m)}$.*

Proof. A proof of this lemma is sketched in [Kra95] and it relies on the bounded arithmetic formal system **PV** (for Polynomially Verifiable) defined by Stephen Cook in [Coo75].

For more details we refer the reader to [Coo75], chapter 4 in [Kra95], and Lemma 9.2.4 in [Kra95]. □

The following two lemmas are used in the proof of Theorem 4.5.6. To make this proof somewhat easier to read, we chose to extract those two parts and add them as lemmas. However, in these lemmas, as well as in the theorem, we follow the proof provided by Krajíček in [Kra95], while trying to add additional details to better understand the proof. The original proof by Krajíček does not include the derivation of each used sequent, which is what we tried to add. However, in our extended proof, we still use some sequents (four in total) without a derivation. This fact will be mentioned every time such a sequent is used in the proof.

Lemma 4.5.4. *If the sequent $[[\Gamma]], [[A(a)]] \longrightarrow [[A(a+1)]], [[\Delta]]$ has a polynomial-time constructible* KPG$_i$*-proof for some Σ_i^b-formula A, then, for each $i \in \{1, \ldots, q(m)\}$, the sequent*

$$[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^i)]](\overline{p}), [[\Delta]] \qquad \cdots\cdots\ S_i$$

has a polynomial-time KPG$_i$*-proof.*

Proof. Since we already have a proof of S_0 (i.e., $[[\Gamma]], [[A(a)]] \longrightarrow [[A(a+1)]], [[\Delta]]$) by assumption, from the sequent S_{i-1} (i.e., $[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^{i-1})]](\overline{p}), [[\Delta]]$) we can derive the sequent S_i for each $i > 0$ as follows.

From the equality axioms and Lemma 4.4.1 we get

$$[[a+2^{i-1}=b]](\overline{p},\overline{q}), [[A(a+2^{i-1})]](\overline{p}) \longrightarrow [[A(b)]](\overline{q}), \qquad \cdots\cdots\ (1)$$

such that $\overline{p}$ are the atoms associated with a, and $\overline{q}$ are the atoms associated with the variable b.

By applying cut on S_{i-1} and (1), we obtain

$$[[\Gamma]], [[a+2^{i-1}=b]](\overline{p},\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(b)]](\overline{q}), [[\Delta]]. \qquad \cdots\cdots\ (2)$$

By substituting $\overline{p}$ with $\overline{q}$ in S_{i-1}, such that the elements of $\overline{p}$ do not occur in $[[\Gamma]]$ and $[[\Delta]]$, we get

$$[[\Gamma]], [[A(b)]](\overline{q}) \longrightarrow [[A(b+2^{i-1})]](\overline{q}), [[\Delta]]. \qquad \cdots\cdots \ (3)$$

Cut on (2) and (3) yields the sequent

$$[[\Gamma]], [[a+2^{i-1}=b]](\overline{p},\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(b+2^{i-1})]](\overline{q}), [[\Delta]]. \qquad \cdots\cdots \ (4)$$

From the equality axioms and Lemma 4.4.1, we can get the following

$$[[a+2^{i-1}=b]](\overline{p},\overline{q}), [[A(b+2^{i-1})]](\overline{q}) \longrightarrow [[A(a+2^{i})]](\overline{p}). \qquad \cdots\cdots \ (5)$$

By applying cut on (4) and (5) we get

$$[[\Gamma]], [[a+2^{i-1}=b]](\overline{p},\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^{i})]](\overline{p}), [[\Delta]]. \qquad \cdots\cdots \ (6)$$

Then by applying $[\exists l]$ $(q(m)+1)$ times on $\overline{q}$, such that $\overline{q}$ are eigenvariables not occurring free in the lower sequent, we get

$$[[\Gamma]], \exists\overline{x}[[a+2^{i-1}=b]](\overline{p},\overline{x}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^{i})]](\overline{p}), [[\Delta]]. \qquad \cdots\cdots \ (7)$$

We take the sequent $\longrightarrow \exists\overline{x}[[a+2^{i-1}=b]](\overline{p},\overline{x})$ from [Kra95] without a derivation (sequent (V_7) in the original proof by Krajíček). By applying cut on this sequent with (7) we get S_i, which is

$$[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^{i})]](\overline{p}), [[\Delta]]. \qquad \square$$

Lemma 4.5.5. *If the sequent* $[[\Gamma]], [[A(a)]] \longrightarrow [[A(a+1)]], [[\Delta]]$ *for some* Σ_i^b*-formula* A*, has a polynomial-time constructible* KPG$_i$*-proof, then we can construct a* KPG$_i$*-proof of the sequent*

$$[[\Gamma]], [[2^{q(m)} \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p},\overline{q}), [[\Delta]] \qquad \cdots\cdots \ (S_{q(m)})$$

in polynomial-time.

Proof. The idea is similar to the proof of the previous Lemma 4.5.4. By using S_i to denote the sequent $[[\Gamma]], [[2^i \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p},\overline{q}), [[\Delta]]$, we can start with S_0 and then derive S_i from S_{i-1} for each $i \in \{1,\ldots,q(m)\}$

Let $\overline{p}$ be the atoms associated with a, and let $\overline{q}$ be a new set of $q(m)+1$ atoms associated with the variable b. Then we can derive S_0 as follows

$$\dfrac{\dfrac{\text{By assumption} \atop [[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^0)]](\overline{p}), [[\Delta]]}{[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p},\overline{q}), [[\Delta]]}\,[\text{sub}]}{[[\Gamma]], [[2^0 \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p},\overline{q}), [[\Delta]]}\,[\text{Wl}]$$

such that, in the application of the substitution rule we replace the atoms associated with the term 2^0 (i.e., $\overline{2^0}$) by the atoms $\overline{q}$ associated with b, where $\overline{2^0}$ does not occur in $[[\Gamma]]$, and $[[\Delta]]$.

In what follows we derive the sequent S_i for each $i \in \{1, \dots, q(m)\}$ from the sequent S_{i-1} (i.e., $[[\Gamma]], [[2^{i-1} \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]]$).

Let c, d be new variables associated with the atoms $\overline{u}, \overline{v}$.

Applying substitution on S_{i-1}, where $\overline{p}$ and $\overline{q}$ do not occur in $[[\Gamma]]$ and $[[\Delta]]$, we get

$$[[\Gamma]], [[2^{i-1} \geq c]](\overline{u}), [[A(d)]](\overline{v}) \longrightarrow [[A(d+c)]](\overline{v}, \overline{u}), [[\Delta]]. \qquad \cdots\cdots (1)$$

From the equality axioms and Lemma 4.4.1 we have the sequent

$$[[a + 2^{i-1} = d]](\overline{p}, \overline{v}), [[A(a + 2^{i-1})]](\overline{p}) \longrightarrow [[A(d)]](\overline{v}). \qquad \cdots\cdots (2)$$

We have by Lemma 4.5.4 a proof of the sequent $[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+2^{i-1})]](\overline{p}), [[\Delta]]$ for each $i \in \{1, \dots, q(m)\}$. By using this sequent and (2) and applying cut we get

$$[[\Gamma]], [[A(a)]](\overline{p}), [[a + 2^{i-1} = d]](\overline{p}, \overline{v}) \longrightarrow [[A(d)]](\overline{v}), [[\Delta]]. \qquad \cdots\cdots (3)$$

By cut on (1) and (3), we obtain

$$[[\Gamma]], [[2^{i-1} \geq c]](\overline{u}), [[A(a)]](\overline{p}), [[a + 2^{i-1} = d]](\overline{p}, \overline{v}) \longrightarrow [[A(d+c)]](\overline{v}, \overline{u}), [[\Delta]]. \qquad \cdots\cdots (4)$$

From the sequent (4) we derive (as mentioned in [Kra95], specifically the sequent tagged with Z_4 in the proof by Krajíček)

$$[[\Gamma]], [[2^{i-1} \geq c]](\overline{u}), [[A(a)]](\overline{p}), [[b = 2^{i-1} + c]](\overline{q}, \overline{u}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]].$$

We apply $\wedge$l1 and $\wedge$l2, so we get

$$[[\Gamma]], [[(2^{i-1} \geq c) \wedge (b = 2^{i-1} + c)]](\overline{q}, \overline{u}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]]. \qquad \cdots\cdots (5)$$

Apply $\vee$l on S_{i-1} and (5), thus obtaining

$$[[\Gamma]], [[2^{i-1} \geq b \vee (2^{i-1} \geq c \wedge b = 2^{i-1} + c)]](\overline{q}, \overline{u}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]]. \qquad \cdots\cdots (6)$$

After $(q(m) + 1)$ applications of $\exists$l (where $\overline{u}$ are eingenvariables not occurring in the lower sequents), we get

$$[[\Gamma]], \exists\overline{x}[[2^{i-1} \geq b \vee (2^{i-1} \geq c \wedge b = 2^{i-1} + c)]](\overline{q}, \overline{x}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]]. \qquad \cdots\cdots (7)$$

From the sequent $[[2^i \geq b]](\overline{q}) \longrightarrow [[2^i \geq b]](\overline{q})$, we derive

$$[[2^i \geq b]](\overline{q}) \longrightarrow [[2^{i-1} \geq b \vee (2^{i-1} \geq c \wedge b = 2^{i-1} + c)]](\overline{q}, \overline{u}),$$

then with a $(q(m)+1)$ applications of $\exists$r, we obtain

$$[[2^i \geq b]](\overline{q}) \longrightarrow \exists\overline{x}[[2^{i-1} \geq b \vee (2^{i-1} \geq c \wedge b = 2^{i-1} + c)]](\overline{q}, \overline{x}). \qquad \cdots\cdots \ (8)$$

Finally by applying cut on (7) and (8) we get S_i, which is

$$[[\Gamma]], [[2^i \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]].$$

□

Theorem 4.5.6 (Theorem 9.2.5 in [Kra95])**.** *Let $i \geq 1$ and A be a Σ_i^b-formula. Assume that A is provable in T_2^i, then there is a bounding polynomial $q(\cdot)$ for A, such that, for all m, formulas $[[A]]_{q(m)}^m$ have a* KPG$_i$*-proofs of size $m^{O(1)}$ (similarly for S_2^i and* KPG$_i^*$*).*

Proof. By Theorem 4.4.2 and Corollary 4.4.2.1, any T_2^i-theorem A has an (LKB+Σ_i^b-IND) proof π, in which all formulas are in $\Sigma_i^b \cup \Pi_i^b$.

The idea of the proof is to translate π to a KPG$_i$-proof of $[[A]]$, using induction on the structure of π.

We note that through out the proof, if $\Gamma = (A_1, \ldots, A_k)$, then we shall use $[[\Gamma]]$ to denote $[[A_1]], \ldots, [[A_k]]$. Moreover, for any i, we shall use π_i to denote the proof of the sequent S_i.

Recall that $X := \{\overline{\epsilon} \in \{0,1\}^* \mid \epsilon_i = 0 \text{ for } i > |q(m)|\}$ and that $q(\cdot)$ is the bounding polynomial of all formulas in π.

Base Case

For the base case we have to consider the BASIC axioms, the equality axioms, and the logical axioms of the form $A \longrightarrow A$. The later two have a straightforward proof, and proofs of the BASIC axioms follow directly from Lemma 4.5.3.

Induction hypothesis

We assume that all the steps in π up to step n can be translated to KPG$_i$-proofs, i.e., the translation of the premise of any rule we might apply in step $n+1$ already has a polynomial-time constructible KPG$_i$-proof.

Induction Step

Step $n+1$ in π could fall in one of the following cases.

Case 1. The case of the structural, propositional, cut or the unbounded quantification rules, the corresponding rules of KPG$_i$ are used.

The non-trivial cases are the introduction of the bounded quantifiers and the induction rule.

Case 2. Let us consider the case [$\forall \leq$r]:

$$[\forall \leq \mathrm{r}]\ \frac{a \leq s,\ \Gamma \longrightarrow \Delta, A(a)}{\Gamma \longrightarrow \Delta, \forall x \leq s\ A(x)}$$

By the induction hypothesis we already have a KPG$_i$-proof for the translation of the top sequent, from which we shall derive the translation of the bottom sequent. We have two subcases.

Subcase 2.1. The term s is of the form $|t|$ for some term t.

From the equality axioms we can derive for each $\bar{\epsilon} \in X$

$$[[a = b]](\bar{q}/\bar{\epsilon}), [[b \leq |t|]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a \leq |t|]].$$

From this and by applying the rules [$\wedge$l1], and [$\wedge$l2] and the appropriate structural rules we get

$$[[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a \leq |t|]],$$

for each $\bar{\epsilon} \in X$.

Now, by repeatedly applying [$\vee$l], we get

$$\bigvee_{\bar{\epsilon} \in X} ([[a = b \wedge b \leq |t|\ ]](\bar{q}/\bar{\epsilon})) \longrightarrow [[a \leq |t|]]. \qquad \cdots\cdots\ S_1$$

Now consider the following proof, in which a double line means omitted structural rules:

$$[\wedge\mathrm{r}]\ \dfrac{[\supset\mathrm{r}]\ \dfrac{[\supset\mathrm{l}]\ \dfrac{[\vee\mathrm{r}]\ \dfrac{A \longrightarrow A}{A \longrightarrow C, A \vee B} \qquad C \longrightarrow C}{A, (A \vee B) \supset C \longrightarrow C}}{(A \vee B) \supset C \longrightarrow A \supset C} \qquad \dfrac{\text{Similar to the left side}}{(A \vee B) \supset C \longrightarrow B \supset C}}{(A \vee B) \supset C \longrightarrow (A \supset C) \wedge (B \supset C)}$$

This proof can be generalized, thus starting from

$$[[a = b \wedge b \leq |t|\ ]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a = b \wedge b \leq |t|\ ]](\bar{q}/\bar{\epsilon}) \qquad \text{for every } \bar{\epsilon} \in X$$

and $[[A(a)]] \longrightarrow [[A(a)]]$ we derive the sequent S_2, which is the following.

$$\Big(\bigvee_{\bar{\epsilon} \in X} [[a = b\ \wedge\ b \leq |t|]](\bar{q}/\bar{\epsilon}) \Big) \supset [[A(a)]] \longrightarrow \bigwedge_{\bar{\epsilon} \in X} [[a = b\ \wedge\ b \leq |t| \supset A(a)]](\bar{q}/\bar{\epsilon}).$$

From S_1 we can then construct a proof of the sequent S_3 as follows:

$$\dfrac{\dfrac{\begin{matrix}\pi_1 \\ \vdots \\ S_1\end{matrix} \qquad [[A(a)]] \longrightarrow [[A(a)]]}{\bigvee_{\bar{\epsilon} \in X} ([[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}))\, , [[a \leq |t|]] \supset [[A(a)]] \longrightarrow [[A(a)]]}\,[\supset\mathrm{l}]}{[[a \leq |t|]] \supset [[A(a)]] \longrightarrow (\bigvee_{\bar{\epsilon} \in X} ([[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}))) \supset [[A(a)]] \quad \cdots\ S_3}\,[\supset\mathrm{r}]$$

For each $\bar{\epsilon} \in X$ we derive the sequent S_ϵ as follows. Starting from the sequents

$$[[a = b]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a = b]](\bar{q}/\bar{\epsilon}) \quad \text{and} \quad [[b \leq |t|]](\bar{q}/\bar{\epsilon}) \longrightarrow [[b \leq |t|]](\bar{q}/\bar{\epsilon}),$$

by applying $\wedge$r with the appropriate structural rules, we can derive

$$[[a = b]](\bar{q}/\bar{\epsilon}), [[b \leq |t|]](\bar{q}/\bar{\epsilon}) \longrightarrow [[A(a)]], [[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}).$$

By applying $\supset$l on this sequent and the sequent $[[A(a)]] \longrightarrow [[A(a)]]$ (with the appropriate structural rules), we get

$$[[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}) \supset [[A(a)]], [[a = b]](\bar{q}/\bar{\epsilon}), [[b \leq |t|]](\bar{q}/\bar{\epsilon}) \longrightarrow [[A(a)]].$$

After two application of $\supset$r rule, we get

$$[[a = b \wedge b \leq |t|]](\bar{q}/\bar{\epsilon}) \supset [[A(a)]] \longrightarrow [[a = b \supset (b \leq |t| \supset A(a))]](\bar{q}/\bar{\epsilon}).$$

From Lemma 4.5.1 we have

$$[[a = b \supset (b \leq |t| \supset A(a))]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a \leq |t| \supset A(a)]](\bar{p}/\bar{\epsilon}).$$

By applying cut on $[[a = b \supset (b \leq |t| \supset A(a))]](\bar{q}/\bar{\epsilon})$, and repeated application of $\wedge$l we obtain

$$\bigwedge_{\bar{\epsilon} \in X} [[a = b \wedge b \leq |t| \supset A(a)]](\bar{q}/\bar{\epsilon}) \longrightarrow [[a \leq |t| \supset A(a)]](\bar{p}/\bar{\epsilon}). \qquad \cdots\cdots\ S_\epsilon$$

We then drive S_4 after repeated application of $\wedge$r on the sequents S_ϵ (one for each $\bar{\epsilon} \in X$) and obtain

$$\bigwedge_{\bar{\epsilon} \in X} [[a = b \wedge b \leq |t| \supset A(a)]](\bar{q}/\bar{\epsilon}) \longrightarrow \bigwedge_{\bar{\epsilon} \in X} [[a \leq |t| \supset A(a)]](\bar{p}/\bar{\epsilon}). \qquad \cdots\cdots\ S_4$$

Now from S_3, S_2, and S_4 we prove S_5 as follows

$$\dfrac{\dfrac{\begin{array}{c}\pi_3\\ \vdots\\ S_3\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ S_2\end{array}}{[[a \leq |t|\ ]] \supset [[A(a)]] \longrightarrow \bigwedge_{\bar{\epsilon} \in X} [[a = b \wedge b \leq |t| \supset A(a)]](\bar{q}/\bar{\epsilon})}[\text{cut}] \qquad \begin{array}{c}\pi_4\\ \vdots\\ S_4\end{array}}{[[a \leq |t|\ ]] \supset [[A(a)]] \longrightarrow \bigwedge_{\bar{\epsilon} \in X} [[a \leq |t| \supset A(a)]](\bar{p}/\bar{\epsilon})}[\text{cut}] \qquad \cdots\ S_5$$

Now from the induction hypothesis the upper sequent of $\forall \leq$r has a proof, then we can with an application of $\supset$r we get S_6 as follows.

$$[\supset\text{r}]\ \dfrac{[[a \leq |t|\ ]],\ [[\Gamma]] \longrightarrow [[\Delta]], [[A(a)]]}{[[\Gamma]] \longrightarrow [[\Delta]], [[a \leq |t|\ ]] \supset [[A(a)]]}$$

Finally by applying cut on S_6 and S_5 we get the sequent

$$[[\Gamma]] \longrightarrow [[\Delta]], \bigwedge_{\overline{\epsilon}\in X} [[a \leq |t| \supset A(a)]](\overline{p}/\overline{\epsilon}).$$

Thus we now have a KPG$_i$-derivation of the translation of $\forall x \leq sA(x)$.

Subcase 2.2. s is not of the form $|t|$

In this case, from the translation of the upper sequent of the $\forall \leq$r, we get

$$[\supset\text{r}]\ \frac{[[a \leq s\ ]],\ [[\Gamma]] \longrightarrow [[\Delta]], [[A(a)]]}{[[\Gamma]] \longrightarrow [[\Delta]], [[a \leq s\ ]] \supset [[A(a)]]}$$

After $q(m)+1$ application of the [$\forall$r] rule to $\overline{p}$, where the elements of $\overline{p}$ are eigenvariables not occurring in the lower sequents after applying the rule, we get the end-sequent we want

$$[[\Gamma]] \longrightarrow [[\Delta]], \forall x_0, \ldots, x_{q(m)}[[a \leq s \supset A(a)]](\overline{p}/\overline{x})$$

which is, by the definition of the translation, the same as

$$[[\Gamma]] \longrightarrow [[\Delta]], [[\forall x \leq sA(x)]].$$

Case 3. [$\forall \leq$l]:

$$[\forall \leq\text{l}]\ \frac{A(t), \Gamma \longrightarrow \Delta}{t \leq s,\ \forall x \leq s\ A(x), \Gamma \longrightarrow \Delta}$$

By the induction hypothesis we already have a KPG$_i$-proof for the translation of the top sequent, from which we shall derive the translation of the bottom sequent. We have again two sub-cases.

Subcase 3.1. The term s is of the form $|r|$, for some term r.

Recall from the definition of the translation that the following holds

$$[[\forall x \leq s\ A(x)]] = \bigwedge_{\overline{\epsilon}\in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}).$$

Without a derivation we take the first sequent from [Kra95] (also tagged with S_1 in the proof by Krajíček)

$$[[t \leq |r|\ ]] \longrightarrow \bigvee_{\overline{\epsilon}\in X} [[t = a \wedge a \leq |r|\ ]](\overline{p}/\overline{\epsilon}). \qquad \cdots\cdots\ S_1$$

For each $\overline{\epsilon} \in X$, we derive the sequent S_ϵ as follows. From the sequents

$$[[a \leq |r|]](\overline{p}/\overline{\epsilon}) \longrightarrow [[a \leq |r|]](\overline{p}/\overline{\epsilon}) \qquad \text{and} \qquad [[A(a)]](\overline{p}/\overline{\epsilon}) \longrightarrow [[A(a)]](\overline{p}/\overline{\epsilon}),$$

by applying $\supset$l with the appropriate structural rules, we get

$$[[t = a]](\overline{p}/\overline{\epsilon}), [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}), [[a \leq |r|]](\overline{p}/\overline{\epsilon}) \longrightarrow [[A(a)]](\overline{p}/\overline{\epsilon}).$$

By applying $\wedge$r on this sequent and the sequent $[[t=a]](\overline{p}/\overline{\epsilon}) \longrightarrow [[t=a]](\overline{p}/\overline{\epsilon})$ (with the appropriate structural rules), we obtain

$$[[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}), [[t=a]](\overline{p}/\overline{\epsilon}), [[a \leq |r|]](\overline{p}/\overline{\epsilon}) \longrightarrow [[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}).$$

After reapeated application of $\wedge$l, and $\vee$r, we get

$$\bigwedge_{\overline{\epsilon} \in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}), [[t = a \wedge a \leq |r|]](\overline{p}/\overline{\epsilon}) \longrightarrow \bigvee_{\overline{\epsilon} \in X} [[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}). \qquad \cdots\cdots\ S_\epsilon$$

By applying $\vee$l on the sequents S_ϵ (a sequent for each $\overline{\epsilon} \in X$), we get the following sequent

$$\bigwedge_{\overline{\epsilon} \in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}), \bigvee_{\overline{\epsilon} \in X} [[t = a \wedge a \leq |r|\ ]](\overline{p}/\overline{\epsilon}) \longrightarrow \bigvee_{\overline{\epsilon} \in X} [[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}) \qquad \cdots\cdots\ S_2$$

and by cut on S_1 and S_2, we obtain

$$[[t \leq |r|\ ]], \bigwedge_{\overline{\epsilon} \in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}) \longrightarrow \bigvee_{\overline{\epsilon} \in X} [[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}). \qquad \cdots\cdots\ S_3$$

From Lemma 4.5.2 we have, for each $\overline{\epsilon} \in X$, $[[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}) \longrightarrow [[A(t)]]$. By applying [$\vee$l] on these sequents we get

$$\bigvee_{\overline{\epsilon} \in X} [[t = a \wedge A(a)]](\overline{p}/\overline{\epsilon}) \longrightarrow [[A(t)]]. \qquad \cdots\cdots\ S_4$$

By applying the cut rule on S_3 and S_4, we get

$$[[t \leq |r|]], \bigwedge_{\overline{\epsilon} \in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}) \longrightarrow [[A(t)]]. \qquad \cdots\cdots\ S_5$$

By the induction hypothesis we have a proof of the upper sequent of the [$\forall \leq$l] rule, i.e., $[[A(t)]], [[\Gamma]] \longrightarrow [[\Delta]]$ to which with S_5 we apply the cut rule to get the sequent

$$[[t \leq |r|]], \bigwedge_{\overline{\epsilon} \in X} [[a \leq |r| \supset A(a)]](\overline{p}/\overline{\epsilon}), [[\Gamma]] \longrightarrow [[\Delta]].$$

Subcase 3.2. The term s is not of the form $|r|$.
Recall from the definition of the translation that the following holds.

$$[[\forall x \leq s\ A(x)]] = \forall \overline{x}\ [[a \leq s \supset A(a)]](\overline{p} \backslash \overline{x})$$

We first take the sequent

$$[[t \leq s]] \longrightarrow \exists \overline{x}\ [[a \leq s \wedge a = t]](\overline{p}/\overline{x}) \qquad \cdots\cdots\ S_1$$

without a derivation from [Kra95] (also tagged with S_1 in the proof by Krajíček). To get S_2, we first derive the following.

$$\dfrac{\dfrac{[[a \le s]] \longrightarrow [[a \le s]]}{[[a \le s]] \longrightarrow A(a), [[a \le s]]} \qquad \dfrac{[[A(a)]] \longrightarrow [[A(a)]]}{[[A(a)]], [[a \le s]] \longrightarrow [[A(a)]]}}{[[a \le s \supset A(a)]], [[a \le s]] \longrightarrow [[A(a)]]}\,[\supset\mathrm{l}]$$

After an application of $\wedge$r (with the appropriate structural rules) on this sequent and the sequent $[[a = t]] \longrightarrow [[a = t]]$ we get

$$[[a \le s \supset A(a)]], [[a \le s]], [[a = t]] \longrightarrow [[A(a) \wedge a = t]].$$

By applying $\wedge$l, we obtain

$$[[a \le s \supset A(a)]], [[a \le s \wedge a = t]] \longrightarrow [[A(a) \wedge a = t]].$$

We then get S_2 as follows.

$$[\exists\mathrm{l}]\,\dfrac{[\exists\mathrm{r}]\,\dfrac{[\forall\mathrm{l}]\,\dfrac{[[a \le s \supset A(a)]], [[a \le s \wedge a = t]] \longrightarrow [[A(a) \wedge a = t]]}{\forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}), [[a \le s \wedge a = t]] \longrightarrow [[A(a) \wedge a = t]]}}{\forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}), [[a \le s \wedge a = t]] \longrightarrow \exists\overline{x}[[A(a) \wedge a = t]](\overline{p}\backslash\overline{x})}}{\forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}), \exists\overline{x}[[a \le s \wedge a = t]](\overline{p}\backslash\overline{x}) \longrightarrow \exists\overline{x}[[A(a) \wedge a = t]](\overline{p}\backslash\overline{x})}$$

The elements of $\overline{p}$ in the application of $\exists$l are eigenvariables not occurring in the lower sequent.
By cut on S_1 and S_2 we get

$$[[t \le s]], \forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}) \longrightarrow \exists\overline{x}[[a = t \wedge A(a)]](\overline{p}/\overline{x}). \qquad \cdots\cdots\ S_3$$

Again from Lemma 4.5.2, we have $[[t = a \wedge A(a)]] \longrightarrow [[A(t)]]$ from which we derive the sequent

$$\exists\overline{x}[[t = a \wedge A(a)]](\overline{p}/\overline{x}) \longrightarrow [[A(t)]] \qquad \cdots\cdots\ S_4$$

where the elements of $\overline{p}$ are eigenvariables not occurring in the lower sequents after applying the $\exists$l rules.
Just like in the previous case, by the cut rule on S_3 and S_4, we obtain

$$[[t \le s]], \forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}) \longrightarrow [[A(t)]]. \qquad \cdots\cdots\ S_5$$

By the induction hypothesis we have a proof of the upper sequent of the $[\forall \le \mathrm{l}]$ rule, i.e., $[[A(t)]], [[\Gamma]] \longrightarrow [[\Delta]]$, to which with S_5 we apply the cut rule to get the sequent

$$[[t \le s]], \forall\overline{x}[[a \le s \supset A(a)]](\overline{p}\backslash\overline{x}), [[\Gamma]] \longrightarrow [[\Delta]].$$

Case 4. The cases of the bounded existential rules are the duals of the universal cases.

Case 5. Σ_i^b-IND rule:

$$\dfrac{\Gamma, A(a) \longrightarrow A(a+1), \Delta}{\Gamma, A(0) \longrightarrow A(t), \Delta}$$

One possibility to simulate the induction rule is by applying cuts on the upper sequents for $a = 0, \ \ldots, \ t-1$, however this would be of an exponential size. As an alternative we shall use the substitution rule, which we presented above before Lemma 4.5.4.

Since by the induction hypothesis we have a polynomial-time constructible KPG$_i$-proof of the sequent

$$[[\Gamma]], [[A(a)]](\overline{p}) \longrightarrow [[A(a+1)]](\overline{p}, \overline{q}), [[\Delta]],$$

we can derive by Lemma 4.5.5 the sequent

$$[[\Gamma]], [[2^{q(m)} \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]].$$

By the definition of the bounding polynomial $q(\cdot)$, we have $|t| \leq q(m)$, which entails that $t \leq 2^{q(m)}$ is true. Thus $[[t \leq 2^{q(m)}]]$ is true by the construction of the translation, and a simulation of the IND-rule in KPG$_i$ is as follows

$$[\text{cut}]\ \dfrac{\longrightarrow [[2^{q(m)} \geq t]](\overline{r}) \qquad [\text{sub}]\ \dfrac{[[\Gamma]], [[2^{q(m)} \geq b]](\overline{q}), [[A(a)]](\overline{p}) \longrightarrow [[A(a+b)]](\overline{p}, \overline{q}), [[\Delta]]}{[[\Gamma]], [[2^{q(m)} \geq t]](\overline{r}), [[A(0)]] \longrightarrow [[A(t)]](\overline{r}), [[\Delta]]}}{[[\Gamma]], [[A(0)]] \longrightarrow [[A(t)]](\overline{r}), [[\Delta]]}$$

such that, in the application of [sub], we are substituting $\overline{p}$ with $\overline{0}$, and $\overline{q}$ with $\overline{r}$, where $\overline{p}$ and $\overline{q}$ do not occur in $[[\Gamma]]$ and $[[\Delta]]$. □

We learned so far that we can translate formulas from bounded arithmetic to QBFs, we can also transform a proof of a bounded formula to a KPG-proof, and subsequently a G-proof, of a related QBF. Cook and Morioka in [CM05] (also Morioka in [Mor05]) were able to utilize this strong connection between QBFs and bounded arithmetic to build on exiting bounded arithmetic results to prove new complexity results for the Σ_1^q-Witnessing problem.

In the following chapters we focus mainly on G_0 and G_0^* and the complexity of the witnessing problem for these systems.

CHAPTER 5

First-order Results for G_0 and G_0^*

As mentioned earlier the modifications introduced by Cook and Morioka in [CM05] give us complete proof systems G_i, G_i^*, which can reason about any QBF, and have a correspondence with the theories of bounded arithmetic T_2^i and S_2^i. By definition, the systems G_i differ from each other by the complexity of the cut formula, and the same holds for the systems G_i^*. This implies that these system are concerned about the complexity of the proofs, rather than the existence of a proof unlike the theories of bounded arithmetic.

Note that KPG_0 and KPG_0^* are quantifier-free propositional proof systems i.e., PK. However G_0 and G_0^* are new systems for QBFs, in which the cut formula of the cut rule is restricted to a quantifier-free formula. These systems are similar to first-order theories T axiomatized by purely universal formulas, each of which has an LK-proof with all cuts on quantifier-free formulas. This result has been proven by Buss in [Bus98].

For the above reasons it seems reasonable to see how applicable are the proof-theoretic results regarding T to G_0 and G_0^*.

5.1 Herbrand Theorem

Herbrand's fundamental theorem describes a relation between predicate and propositional logic, and it basically relates the validity of a first-order formula with the validity of a finite set of propositional formulas. This was an important first step toward automated deduction in first-order logic.

Intuitively the idea is that a prenex first-order formula φ with existential quantifiers is valid if and only if a disjunction of a finite set of instances of the subformulas of φ is

valid.
To prove his theorem Herbrand first transforms φ, while preserving validity, to a formula φ_s in prenex normal form with existential quantifiers only through Herbrandization (a more general case of Skolemization which only preserves satisfiability), which then he used to prove the statement above.

Definition 5.1.1 (π-*prototype*, Herbrand π-disjunction)
Let π be a G_0-proof with end-sequent $\longrightarrow \varphi$, where φ is a QBF in prenex form. Then any quantifier-free formula φ' in π that occurs as the auxiliary formula of a quantifier introduction step is called a π*-prototype* of φ.

We define the *Herbrand* π*-disjunction* to be the sequent $\longrightarrow \varphi_1, \ldots, \varphi_m$ where $\varphi_1, \ldots, \varphi_m$ are all the π-prototypes of φ. ◁

Example 5.1
Consider the following G_i^*-proof of the formula $\longrightarrow \forall x \exists y (x \vee y) \wedge (\neg x \vee y)$.

$$\text{[}\forall\text{r]}\ \dfrac{\text{[}\exists\text{r]}\ \dfrac{\text{[}\wedge\text{r]}\ \dfrac{\text{[}\vee\text{r2]}\ \dfrac{\longrightarrow \top}{\longrightarrow \neg p \vee \top} \qquad \dfrac{\longrightarrow \top}{\longrightarrow p \vee \top}\ \text{[}\vee\text{r2]}}{\longrightarrow (p \vee \top) \wedge (\neg p \vee \top)}}{\longrightarrow \exists y (p \vee y) \wedge (\neg p \vee y)}}{\longrightarrow \forall x \exists y (x \vee y) \wedge (\neg x \vee y)}$$

The formula $(p \vee \top) \wedge (\neg p \vee \top)$ is the only π-prototype in this proof. Therefore the Herbrand π-disjunction is the sequent $\longrightarrow (p \vee \top) \wedge (\neg p \vee \top)$.

Definition 5.1.2
Let π be a G_0-proof of φ, where φ is a QBF in prenex normal form, i.e., it is of the form

$$Q_1 x_1 \cdots Q_k x_k\ \psi(\overline{p}, x_1, \ldots, x_k)$$

such that $\overline{p}$ is a sequence of free variables, $Q_i \in \{\exists, \forall\}$ for each $i \in \{1, \ldots, k\}$, and ψ is quantifier-free.

Then, for each π-prototype φ_j of φ, there exists a unique sequence $\varphi_1^j, \ldots, \varphi_k^j$ of propositional formulas such that

$$\varphi_j = \psi(\overline{p}, \varphi_1^j, \ldots, \varphi_k^j).$$

We call φ_i^j *the ith component of* φ_j. ◁

The following is the G_0 form of Herbrand's theorem for first-order logic. More details and a proof of the theorem for first-order logic can be found in [Bus98].

We note that the proof of this lemma depends on the fact that the formula to be proven is in prenex normal form, and all cut formulas in G_0-proofs are quantifier-free, i.e., this particular proof does not work with a different proof system which allows more complex cut formulas.

Lemma 5.1.1. *Let π be a G_0-proof of a sequent $\longrightarrow \varphi$ such that φ is a QBF in prenex form. Then the Herbrand π-disjunction is valid, and it has a* PK*-proof of size polynomial in $|\pi|$.*

Proof. This is a more detailed version of the proof of Lemma 3 in [CM05] and Lemma 5.25 in [Mor05]

Assume that π is the sequence $S_1, \ldots, S_k$ of sequents such that S_k is the sequent $\longrightarrow \varphi$.

For every $i \in \{1, \ldots, k\}$, if sequent S_i is $\Gamma_i \longrightarrow \Delta_i$, then we define the sequent S_i' to be $\Gamma_i \longrightarrow \Delta_i'$ such that

- If Δ_i has no quantified formulas then $\Delta_i = \Delta_i'$,
- else Δ_i' is obtained by removing all quantified formulas from Δ_i and adding all π-prototypes $\varphi_1, \ldots, \varphi_m$.

Note that the following statements hold.

1. S_k' is the Herbrand π-disjunction.
2. Because π is a G_0-proof of a QBF in prenex normal form, and cut is only allowed on quantifier-free formulas, then all formulas in Γ_i must be quantifier-free, for any $i \geq 0$.
3. If all formulas in the sequent S_i are quantifier-free, then $S_i' = S_i$

We prove that S_i' (and eventually S_k') has a PK-proof of size polynomial in $|\pi|$. The proof is performed by strong induction on i.

Base case: $i = 1$.

The initial sequents in G_0 can be one of these three sequents:

$$\longrightarrow \top, \qquad \bot \longrightarrow \qquad \text{or} \qquad \psi \longrightarrow \psi$$

where ψ can be any QBF.

If π uses the sequent $\psi \longrightarrow \psi$, then ψ must be quantifier-free, because of the reason mentioned in point 2. above.

This means that all three possible initial sequents are quantifier-free, thus $S_1 = S_1'$, and no additional proof is needed to derive S_1'.

Induction hypothesis

We assume that for some $i \geq 1$ and all $j \leq i$, S_j' has a PK-proof of size polynomial in $|\pi|$.

Induction step

If S_{i+1} does not have a quantified formula then $S_{i+1} = S'_{i+1}$ and no additional proof is needed.

If S_{i+1} has a quantified formula, then this formula can only be in Δ_{i+1} (note the statement 2. above). The only non-trivial case is when S_{i+1} is derived from S_j, for some $j \leq i$, which is quantifier-free, i.e., $S_j = S'_j$. In this case either the weakening or a quantifier introduction rule was applied on S_j (both on the right side).

If weakening was applied, then we start with S'_j, which has by the induction hypothesis a PK-proof of polynomial size, and then we introduce $\varphi_1, \dots, \varphi_m$ by applying the weakening on the right side of S'_j.

If a quantifier introduction rule was applied, and φ_k is the formula on which we apply the quantifier rule, we can again apply the weakening to introduce $\varphi_1, \dots, \varphi_{k-1}, \varphi_{k+1}, \dots, \varphi_m$.

Note that in both cases we extended a proof of size polynomial in $|\pi|$, with at most m steps of weakening which implies that the size of the proof is still polynomial in $|\pi|$. □

5.2 The Midsequent Theorem

We first recall the original midsequent theorem for first-order logic, introduced by Gerhard Gentzen in [Gen35]. An English version of the paper is in the book "The Collected Papers of Gerhard Gentzen" [Gen69].

Theorem 5.2.1 (The Midsequent Theorem for LK)**.** *Assume we have an* LK*-derivation of the sequent S, where S consists of prenex formulas only. Then S has a treelike cut-free* LK*-derivation with a "midsequent" S' such that the part of the proof above S' is propositional (i.e., the derivation of S' does not include quantifier rules), and the part of the proof below S' consists only of quantifier and structural rules.*

Krajíček pointed out in [Kra95] that the theorem holds for KPG, but Cook and Morioka in [CM05] and Morioka in [Mor05] proved even a polynomial-time version of it for G_0^*, which is the following.

Theorem 5.2.2 (The polynomial-time Midsequent Theorem for G_0^*)**.** *Let π be a G_0^*-proof of sequent S of the form $\longrightarrow \varphi$, such that φ is a QBF in prenex form. Then there exists a G_0^*-proof π' of S such that*

(i) π' starts with a quantifier-free derivation of the Herbrand π-disjunction S_π, and

(ii) only contraction, exchange, $\forall r$, and $\exists r$ inference steps occur between S_π and the end-sequent.

There is polynomial-time algorithm that converts π to π'.

Proof. Recall that the proof π is in free-variable normal form, as we assume in this thesis that every treelike proof is in this form as mentioned after Definition 3.2.9.

Assume φ is of the form $Q_1 x_1 \dots Q_k x_k \psi(\overline{p}, x_1, \dots, x_k)$ such that $Q_i \in \{\exists, \forall\}$ for each $i \in \{1, \dots, k\}$ and ψ is quantifier-free. Let $\varphi_1, \dots, \varphi_m$ be all the π-prototypes, and let $\varphi_1^j, \dots, \varphi_k^j$ be the components of φ_j.

Since $\varphi_j = \psi(\overline{p}, \varphi_1^j, \dots, \varphi_k^j)$, each φ_i^j will be in π the target of the $\exists$r rule or the eigenvariable of the $\forall$r rule in the steps that introduce the bounded variable x_i into φ_j. It is possible for such a step to be associated with more than one component say $\varphi_i^{j_1}, \dots \varphi_i^{j_c}$, however this happens only if the descendants of the prototypes $\varphi_{j_1}, \dots, \varphi_{j_c}$ are contracted to form one formula at some point in π before the inference introducing the quantifier is applied.

By Lemma 5.1.1 the Herbrand π-disjunction S_π has a PK-proof of size polynomial in $|\pi|$, which means that it is enough to prove that we can derive $\longrightarrow \varphi$ from S_π in polynomial time.

A polynomial-time algorithm that generates π' from π would derive $\longrightarrow \varphi$ from S_π by trying to transform each φ_j into a copy of φ with the rules of contraction, exchange, $\forall$r and $\exists$r only, and can be described as follows.

We shall use Q-inference to denote a quantifier introduction step or a contraction step of two quantified formulas.
Starting from $\longrightarrow S_\pi$ we apply a Q-inference only after applying all the Q-inferences that precedes it in π, and during which the exchange rule is applied when needed. We know that this sequence of steps exists because π exists. And by contradiction we can prove that the eigenvariable condition in the $\forall$r steps is not violated.

Assuming that this condition is violated at some point, say when applying the rule on the variable $b_i^j = b$ (in component φ_i^j), we get the following.

- There is another formula in the sequent, in which b occurs as some variable b_v^u (in component φ_v^u), and
- the elimination step of b_v^u does not precede that of b_i^j in π.

This implies that if π_b is the subproof of π which has the elimination step of b_i^j as an end-sequent, then the elimination step of b_v^u occurs outside π_b, but this contradicts the fact that π is in free-variable normal form. As a consequence, the eigenvariable condition can never be violated. □

As pointed out in [CM05], this proof does not work if π is not treelike, because this means that we can not argue with the free-variable normal form, as this is a form of treelike proofs only (see Definition 3.2.9). Thus, in this case, there might be two $\forall$r steps on two formulas ψ_1 and ψ_2 with the same eigenvariable b such that neither precedes the other in π. In this case when we try to derive $\longrightarrow \varphi$ from S_π we will have at some point both ψ_1 and ψ_2 in the same sequent which means it will not be possible to apply the $\forall$r rule on any of them. However if all quantifier introduction rules in π are $\exists$r then this will no longer be an issue, which is proven in the following theorem.

Theorem 5.2.3. *Theorem 5.2.2 holds for G_0 if the end-formula is prenex Σ_1^q.*

Proof. The proof is taken from [CM05]

Since the end-formula φ is Σ_1^q, any quantifier introduction step would be $\exists$r. Due to the fact that $\forall$r is not needed here, we do not need to worry about the eigenvariable condition.

Then we can easily derive a sequent containing m copies of φ from the Herbrand π-disjunction $\longrightarrow \varphi_1, \dots, \varphi_m$, by a repeated application of the $\exists$r rule. □

Morioka in fact shows in his dissertation [Mor05] that the short PK-proof of the Herbrand π-disjunction as well as π' from the midsequent theorem are both in $\mathbf{TC}^0$.

Krajíček in [Kra95] proves that treelike PK p-simulates PK, which means that G_0^* p-simulates G_0 for propositional formulas. For a general p-simulation (without restrictions on the formulas) the eigenvariables are a major problem.

While we do not have a proof for a general p-simulation, however Morioka in [Mor05] provides a stronger result than that of Krajíček, stated in the following theorem.

Theorem 5.2.4. *G_0^* p-simulates G_0 for proving prenex Σ_1^q-formulas.*

Proof. The proof is taken from [CM05]

Let π be a G_0-proof of a sequent S containing one prenex Σ_1^q-formula. By Theorem 5.2.3 there exists a proof π' of S from the Herbrand π-disjunction S_π.

The proof π' is composed of two parts $\pi_1\pi_2$, such that π_1 is a PK-proof rooted at S_π, and π_2 which is basically repeated application of $\exists$r and contraction.

Since treelike PK p-simulates PK, then we can get a G_0^*-proof of S by replacing π_1 in π' to a treelike version of it with only polynomial size increase. □

CHAPTER 6

The Witnessing Problem for G_0 and G_0^*

We now have the tools we need to take a closer look at the complexity of the witnessing problem for the calculi G. We focus in this chapter on the complexity results for the systems G_0 and G_0^*, but we will mention the complexity results for G_i to complete the picture.

We start by defining the witnessing problem.

Definition 6.0.1
Let $i \geq 0$ and let H be either G_i or G_i^*.

For $j \geq 1$, define the Σ_j^q-Witnessing problem for H, written Witness$[H, \Sigma_j^q]$, as follows:

The input is $(\pi, \overline{v})$, where π is an H-proof of a Σ_j^q-QBF $\varphi(\overline{p})$ of the form

$$\varphi(\overline{p}) = \exists x_1 \cdots \exists x_k \ \psi(\overline{p}, x_1, \ldots, x_k)$$

with ψ prenex Π_{j-1}^q, and $\overline{v}$ is a truth assignment to the free variables $\overline{p}$.

A solution to the problem is a witness for $\varphi(\overline{v})$, i.e., a truth assignment $\overline{u}$ to the variables $\overline{x}$ such that $\psi(\overline{v}, \overline{u})$ is true. ◁

Theorem 6.0.1. *Witness$[G_1^*, \Sigma_1^q]$ and Witness$[G_1, \Sigma_1^q]$ are complete for* **FP** *and* **PLS**, *respectively. More generally, for $i \geq 1$, Witness$[G_i^*, \Sigma_i^q]$ and Witness$[G_i, \Sigma_i^q]$ are complete for* $\mathbf{FP}^{\Sigma_{i-1}^p}$ *and* $\mathbf{PLS}^{\Sigma_{i-1}^p}$, *respectively.*

Proof. For a proof of this theorem we refer the reader to Theorem 7 in [CM05] or Theorem 6.2 in [Mor05] □

From this theorem it follows that, if G_1^* p-simulates G_1, then **FP** = **PLS**.

The systems G_0 and G_0^* were first introduced by Morioka in his dissertation in 2005 [Mor05], in which he studies the complexity of the witnessing problem for those systems. We present those results in the following two sections.

6.1 The Σ_1^q-Witnessing problem for G_0

Recall that G_0 is the sequent-like proof system for QBFs, in which the cut rule is restricted to quantifier-free (propositional) formulas only.

6.1.1 Witness formulas

Definition 6.1.1 (ith π-witness formula)
Let π be a G_0-proof of a Σ_1^q-QBF $\varphi(\overline{p}) = \exists x_1 \cdots \exists x_k \psi(\overline{p}, x_1, \ldots, x_k)$ such that ψ is quantifier-free, and $\overline{p}$ is a sequence of all free variables on φ. Let $\varphi_1, \ldots, \varphi_m$ be all the π-prototypes.

For each $j \in \{1, \ldots, m\}$, we define the formula $\epsilon_j = (\neg\varphi_1 \wedge \neg\varphi_2 \wedge \cdots \wedge \neg\varphi_{j-1}) \wedge \varphi_j$, meaning that ϵ_j is true when φ_j is the first satisfied formula from $\varphi_1, \ldots, \varphi_m$, which entails that for each satisfying assignment only one ϵ_j is satisfied.

For each $i \in \{1, \ldots, k\}$, let

$$\eta_i = \bigvee_{j=1}^{m} (\epsilon_j \wedge \varphi_i^j)$$

such that φ_i^j is the ith component of the π-prototype φ_j.

We call η_i the ith π-witness formula. $\lhd$

Lemma 6.1.1. *Let π be a G_0-proof of $\varphi(\overline{p}) = \exists x_1 \cdots \exists x_k \psi(\overline{p}, x_1, \ldots, x_k)$ such that ψ is quantifier-free, and $\varphi_1, \ldots, \varphi_m$ are all the π-prototypes.*

Then for any $i \in \{1, \ldots, k\}$, the sequent $\epsilon_l \wedge \varphi_i^l, \varphi_j \longrightarrow \varphi_i^j, \varphi_1, \ldots, \varphi_{j-1}$ has a short PK*-proof for each $l, j \in \{1, \ldots, m\}$.*

Proof. Recall that $\epsilon_l = (\neg\varphi_1 \wedge \neg\varphi_2 \wedge \ldots \wedge \neg\varphi_{l-1}) \wedge \varphi_l$, and φ_i^j is the ith component of the π-prototype φ_j

Depending on the value of l, each of these sequents can be proven by one of the following three short PK-proof.

- Case $l = j$.
  We can get the proof starting from the sequent $\varphi_i^j \longrightarrow \varphi_i^j$, by applying the rules exchange, weakening (left and right), and $\wedge$l.

- Case $l < j$.
  The proof in this case starts with $\varphi_l \longrightarrow \varphi_l$, we then get the required sequent by applying the weakening rule left and right, and $\wedge$l repeated $(l-1)$ times to get ϵ_l and then one more to get $\epsilon_l \wedge \varphi_i^l$.
- Case $l > j$.
  This means that the left side of the sequent will have φ_j and $\neg\varphi_j$, thus starting from the axiom $\varphi_j \longrightarrow \varphi_j$ we get, by applying $\neg$l, the sequent $\neg\varphi_j, \varphi_j \longrightarrow$, then just like the previous case, we apply weakening, and $\wedge$l rules multiple times until we get the final sequent. □

The following theorem shows that the π-witness formulas are in fact a solution for Witness$[G_0, \Sigma_1^q]$, i.e, the Σ_1^q-Witnessing problem for G_0, and this fact even has a short PK-proof.

Theorem 6.1.2 (Theorem 8. [CM05]). *Let π be a G_0-proof of φ, a Σ_1^q-QBF in prenex normal form, thus φ is of the form $\exists x_1 \cdots \exists x_k \psi(\overline{p}, x_1, \dots, x_k)$, where ψ is quantifier-free and $\overline{p}$ is a sequence of all free variables in φ.*
Let $\eta_1, \dots, \eta_k$ be the π-witness formulas. Then $\psi(\overline{p}, \eta_1, \dots, \eta_k)$ is a tautology and it has a PK-proof of size polynomial in $|\pi|$.

Proof. Throughout this proof we use a double line in tree proofs to indicate the existence of omitted steps. Those steps could be repeated application of the same rule, applications of the contraction, weakening, or exchange rule, or a combination of both.

For any subformula ψ' of ψ, we prove by structural induction on ψ' that, for any fixed $j \in \{1, \dots, m\}$, the following two sequents have a short PK-proof

$$\psi'[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \psi'[\overline{x}/\overline{\eta}], \varphi_1, \dots, \varphi_{j-1} \qquad (\text{ denoted by } S_{1(j)}^{\psi'})$$

$$\psi'[\overline{x}/\overline{\eta}], \varphi_j \longrightarrow \psi'[\overline{x}/\overline{\varphi}^j], \varphi_1, \dots, \varphi_{j-1} \qquad (\text{ denoted by } S_{2(j)}^{\psi'})$$

such that $\overline{\varphi}^j$ is the sequence of all components of the π-prototype φ_j and $\overline{\eta}$ is the sequence of all π-witness formulas.

First we observe that if ψ' does not contain any occurrence of the variable x, then $\psi' = \psi'[\overline{x}/\overline{\varphi}^j] = \psi'[\overline{x}/\overline{\eta}]$, which means that both sequents are the same. We can get both sequents by applying the weakening and exchange rules on $\psi' \longrightarrow \psi'$.

We utilize in this proof the fact that the proof systems KPG and G allow the axiom $\psi \longrightarrow \psi$ to be any QBF and do not restrict it to atomic formulas only. Though this is not crucial because any $\varphi \longrightarrow \varphi$ has a short proof w.r.t. the complexity of φ.

Base Case Let $\psi' = x_i$ for some $i \in \{1, \dots, k\}$, then $\psi'[\overline{x}/\overline{\varphi}^j] = \varphi_i^j$ and $\psi'[\overline{x}/\overline{\eta}] = \eta_i = \bigvee_{l=1}^{m} (\epsilon_l \wedge \varphi_i^l)$.
In this case a proof of $S_{1(j)}^{\psi'}$ would be the following.

$$
{\scriptstyle[\wedge r]}\;\dfrac{
{\scriptstyle[\wedge r]}\;\dfrac{
{\scriptstyle[\neg r]}\;\dfrac{\varphi_1 \longrightarrow \varphi_1}{\longrightarrow \varphi_1, \neg\varphi_1}
\qquad
\dfrac{\varphi_2 \longrightarrow \varphi_2}{\longrightarrow \varphi_2, \neg\varphi_2}\;{\scriptstyle[\neg r]}
}{\longrightarrow \varphi_1, \varphi_2, \neg\varphi_1 \wedge \neg\varphi_2}
\qquad
\dfrac{\varphi_3 \longrightarrow \varphi_3}{\longrightarrow \varphi_3, \neg\varphi_3}\;{\scriptstyle[\neg r]}
}{\longrightarrow \varphi_1, \varphi_2, \varphi_3, \neg\varphi_1 \wedge \neg\varphi_2 \wedge \neg\varphi_3}
\qquad \vdots
$$

$$
{\scriptstyle[\wedge r]}\;\dfrac{
{\scriptstyle[Wl]}\;\dfrac{
{\scriptstyle[\wedge r]}\;\dfrac{
\dfrac{\varphi_{j-1} \longrightarrow \varphi_{j-1}}{\longrightarrow \varphi_{j-1}, \neg\varphi_{j-1}}\;{\scriptstyle[\neg r]}
}{\longrightarrow \varphi_1, \varphi_2, \ldots, \varphi_{j-1}, \neg\varphi_1 \wedge \neg\varphi_2 \wedge \ldots \wedge \neg\varphi_{j-1}}
}{\varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \neg\varphi_1 \wedge \neg\varphi_2 \wedge \ldots \wedge \neg\varphi_{j-1}}
\qquad
\dfrac{\varphi_j \longrightarrow \varphi_j}{\varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \varphi_j}\;{\scriptstyle[Wr]}
}{\varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \epsilon_j}
$$

$$
{\scriptstyle[\vee r]}\;\dfrac{
{\scriptstyle[\wedge r]}\;\dfrac{\varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \epsilon_j \qquad \varphi_i^j \longrightarrow \varphi_i^j}{\varphi_i^j, \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, (\epsilon_j \wedge \varphi_i^j)}
}{\varphi_i^j, \varphi_j \longrightarrow \bigvee_{l=1}^{m} (\epsilon_l \wedge \varphi_i^l), \varphi_1, \ldots, \varphi_{j-1}}
$$

We can get $S^{\psi'}_{2(j)}$, with $(m-1)$ application of $\vee$r (interleaved with the application of weakening or exchange rules when needed) on the sequents $\epsilon_l \wedge \varphi_i^l, \varphi_j \longrightarrow \varphi_i^j, \varphi_1, \ldots, \varphi_{j-1}$ for each $l \in \{1, \ldots, m\}$. Each of these sequents by Lemma 6.1.1 have a short PK-proof.

Induction hypothesis

We assume that for any ψ'' a subformual of ψ' the sequents $S^{\psi''}_{1(j)}$ and $S^{\psi''}_{2(j)}$ have a short PK-proof.

Induction Step

We distinguish three cases $\psi' = \psi_1 \vee \psi_2$, $\psi' = \psi_1 \wedge \psi_2$, or $\psi' = \neg\psi_1$. The required sequents can be proven from the proofs assumed to exist in the induction hypothesis.

The following are proofs of the sequent $S^{\psi'}_{1(j)}$. The proofs for $S^{\psi'}_{2(j)}$ are analogous.

$\boxed{\psi' = \psi_1 \vee \psi_2}$

$$
{\scriptstyle[Vl]}\;\dfrac{
{\scriptstyle[\vee r]}\;\dfrac{
\dfrac{\begin{array}{c}\text{A proof of } S^{\psi_1}_{1(j)} \text{ exists}\\ \text{by the induction hypothesis}\\ \vdots\end{array}}{\psi_1[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \psi_1[\overline{x}/\overline{\eta}]}
}{\psi_1[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, (\psi_1 \vee \psi_2)[\overline{x}/\overline{\eta}]}
\qquad
\dfrac{
\dfrac{\begin{array}{c}\text{A proof of } S^{\psi_2}_{1(j)} \text{ exists}\\ \text{by the induction hypothesis}\\ \vdots\end{array}}{\psi_2[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, \psi_2[\overline{x}/\overline{\eta}]}
}{\psi_2[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, (\psi_1 \vee \psi_2)[\overline{x}/\overline{\eta}]}\;{\scriptstyle[\vee r]}
}{(\psi_1 \vee \psi_2)[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \ldots, \varphi_{j-1}, (\psi_1 \vee \psi_2)[\overline{x}/\overline{\eta}]}
$$

$\boxed{\psi' = \psi_1 \wedge \psi_2}$

$$
[\wedge\text{r}]\ \dfrac{[\wedge\text{l}]\ \dfrac{\dfrac{\begin{array}{c}\text{A proof of } S^{\psi_1}_{1(j)} \text{ exists}\\ \text{by the induction hypothesis}\\ \vdots\end{array}}{\psi_1[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_1[\overline{x}/\overline{\eta}]}}{(\psi_1 \wedge \psi_2)[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_1[\overline{x}/\overline{\eta}]} \qquad [\wedge\text{l}]\ \dfrac{\dfrac{\begin{array}{c}\text{A proof of } S^{\psi_2}_{1(j)} \text{ exists}\\ \text{by the induction hypothesis}\\ \vdots\end{array}}{\psi_2[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_2[\overline{x}/\overline{\eta}]}}{(\psi_1 \wedge \psi_2)[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_2[\overline{x}/\overline{\eta}]}}{(\psi_1 \wedge \psi_2)[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, (\psi_1 \wedge \psi_2)[\overline{x}/\overline{\eta}]}
$$

$\boxed{\psi' = \neg\psi_1}$

Note that in this case we prove $S^{\psi'}_{1(j)}$ starting from a proof of $S^{\psi_1}_{2(j)}$

$$
[\neg\text{l}]\ \dfrac{[\neg\text{r}]\ \dfrac{\dfrac{\begin{array}{c}\text{A proof of } S^{\psi_1}_{2(j)} \text{ exists}\\ \text{by the induction hypothesis}\\ \vdots\end{array}}{\psi_1[\overline{x}/\overline{\eta}], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_1[\overline{x}/\overline{\varphi}^j]}}{\varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, \psi_1[\overline{x}/\overline{\varphi}^j], (\neg\psi_1)[\overline{x}/\overline{\eta}]}}{(\neg\psi_1)[\overline{x}/\overline{\varphi}^j], \varphi_j \longrightarrow \varphi_1, \dots, \varphi_{j-1}, (\neg\psi_1)[\overline{x}/\overline{\eta}]}
$$

By Definition 5.1.2 $\psi[\overline{x}/\overline{\varphi}^j] = \varphi_j$, thus the above proof implies that if $\psi' = \psi$ and $j = m$ we will have a short PK-proof of the sequent

$$S^{\psi}_{1(m)} = \varphi_m, \varphi_m \longrightarrow \psi[\overline{x}/\overline{\eta}], \varphi_1, \dots, \varphi_{m-1}.$$

On the other hand, by Lemma 5.1.1 the sequent S_π: $\longrightarrow \varphi_1, \dots, \varphi_m$ has a PK-proof of size polynomial in $|\pi|$. This entails that we can construct the following proof

$$
[\text{Cut on } \varphi_m]\ \dfrac{[\text{Wr}]\ \dfrac{\dfrac{\text{proof of } S_\pi \ \vdots}{\longrightarrow \varphi_1, \dots, \varphi_m}}{\longrightarrow \psi[\overline{x}/\overline{\eta}], \varphi_1, \dots, \varphi_m} \qquad \dfrac{\text{proof of } S^{\psi}_{1(m)} \ \vdots}{\varphi_m \longrightarrow \psi[\overline{x}/\overline{\eta}], \varphi_1, \dots, \varphi_{m-1}}}{\longrightarrow \psi[\overline{x}/\overline{\eta}], \varphi_1, \dots, \varphi_{m-1}}
$$

by repeatedly cutting the π-prototypes, by applying the cut rule m times each time on the result of the previous cut and the appropriate $S^{\psi}_{1(j)}$, we will eventually get a proof of $\longrightarrow \psi[\overline{x}/\overline{\eta}]$. □

6.1.2 An NC^1-algorithm for Witness$[G_0, \Sigma_1^q]$

Buss has shown in [Bus87] that the Boolean formula value problem (i.e., evaluating a propositional formula under a given truth assignment) has an **ALogtime** algorithm, and it was proven by Ruzzo in [Ruz79] that **ALogtime** (the class of problems solvable in logarithmic time with an alternating Turing machine) is equivalent to circuits complexity class **ALogtime**-uniform $\mathbf{NC^1}$ and thus equivalent to **DLogtime**-uniform $\mathbf{NC^1}$ by [MBIS90].

This means that evaluating the truth of a Boolean formula given an assignment is in $\mathbf{NC^1}$, and to define an $\mathbf{NC^1}$ algorithm for solving Witness$[G_0, \Sigma_1^q]$, it is enough to find $\mathbf{NC^1}$-functions that can recognize occurrences of the components φ_i^js of the π-prototypes in a G_0-proof π.

This algorithm takes as input an encoded instance of Witness$[G_0, \Sigma_1^q]$, i.e, an encoding of $(\pi, \overline{v})$, and outputs an encoded solution, i.e, an encoding of the π-witness formulas η_i. So to encode G-proofs and QBFs we define the alphabet

$$\Sigma_{\text{QBF}} = \{\top,\ \bot,\ p,\ x,\ 0,\ 1,\ (,\ ),\ \wedge,\ \vee,\ \neg,\ \exists,\ \forall,\ \supset,\ comma,\ \#\},$$

such that,

- *comma* denotes the comma,
- 0 and 1 are used to denote indices, that is, we shall use $p^\frown\overline{i}$ and $x^\frown\overline{i}$ as an encoding of p_i and x_i, in which $^\frown$ denote string concatenation and $\overline{i}$ is the binary representation of i, $\overline{i} \in \{0,1\}^+$,
- the sharp symbol # will be used in the encoding of proofs only (not in formulas), where a proof will be represented as $S_1\#S2\#\cdots\#S_m$, that is, a sequent S_i is either an initial sequent or derived from at most two preceding sequents, and
- we shall use Z to denote a finite string over Σ_{QBF}, and if $Z = \alpha_1\alpha_2\cdots\alpha_n$ then for $1 \leq i \leq j :\ Z[i,j]$ is the substring $\alpha_i\cdots\alpha_j$.

In what follows, we argue that we can write FOM-sentences (thus $\mathbf{TC^0}$ relations) to represent the different predicates needed to construct the aforementioned $\mathbf{NC^1}$ algorithm. In these sentences we use the FOM predicate *Bit* to describe the binary representation of a string Z over Σ_{QBF}.

In [Bus91] Buss proves that parsing a propositional formula (and even a Frege proof) is in $\mathbf{TC^0}$. These results will be extended in what follows to QBFs and G-proofs. Morioka mentions in [Mor05] that the fact this is possible was also known to Buss, and it is possible because counting is possible in $\mathbf{TC^0}$, as he states in his dissertation:

> "We know from our email correspondence with Samuel Buss that this fact has been known to Buss and possibly a few others, but, as far as we know,

> it has not been explicitly stated in print. $\mathbf{NC^1}$ is widely believed to be the smallest complexity class in which counting is possible. Since parsing operations require counting in general, apparently $\mathbf{TC^0}$ is the smallest class containing those parsing operations."

The fact that counting is possible in $\mathbf{TC^0}$ enabled Morioka in [Mor05] to prove a $\mathbf{TC^0}$ version of the midsequent theorem, in which he describes a procedure to recognize treelike proofs (G^*-proofs) if they were encoded with a special representation that relies on brackets. That said, counting is also essential for recognizing formulas in the $\mathbf{NC^1}$-algorithm we aim to describe, because it mainly relies on counting parentheses as well.

We start by defining the notion of an identifier, a tool that will be later used to describe the predicates that represent the quantifier rules.

Definition 6.1.2
Let the symbol $\frown$ denote string concatenation, i.e, $a\frown b = ab$. Let φ be some formula. The identifier of ψ, a subformula of φ, is a finite string over $\{1,2\}$, denoted by $\mathrm{ID}_\varphi(\psi)$ and is defined as follows.

- $\mathrm{ID}_\varphi(\varphi)$ is the empty string ϵ.
- If $\psi = \psi_1 \circ \psi_2$, with $\circ \in \{\wedge, \vee\}$, then $\mathrm{ID}_\varphi(\psi_1) = \mathrm{ID}_\varphi(\psi)\frown 1$ and $\mathrm{ID}_\varphi(\psi_2) = \mathrm{ID}_\varphi(\psi)\frown 2$.
- If $\psi = \neg\psi_1$ or $\psi = Qx\ \psi_1$, with $Q \in \{\forall, \exists\}$, then $\mathrm{ID}_\varphi(\psi_1) = \mathrm{ID}_\varphi(\psi)\frown 1$. $\triangleleft$

Example. If $\varphi = (\forall x\ (\psi_1 \wedge \psi_2)) \vee (\neg\psi_3)$ then:

$\mathrm{ID}_\varphi(\forall x\ (\psi_1 \wedge \psi_2)) = 1$	$\mathrm{ID}_\varphi(\psi_1 \wedge \psi_2) = 11$
$\mathrm{ID}_\varphi(\psi_1) = 111$	$\mathrm{ID}_\varphi(\psi_2) = 112$
$\mathrm{ID}_\varphi(\neg\psi_3) = 2$	$\mathrm{ID}_\varphi(\psi_3) = 21$

Recall from Section 2.2 the following which will be used in the proofs below.

- Theorem 2.2.4 states that

  predicate R is in $\mathbf{TC^0}$ if and only if R can be represented by an FOM formula.
- $\mathbf{TC^0} \subseteq \mathbf{NC^1}$.

Lemma 6.1.3. *The following predicates are in* $\mathbf{TC}^0$*:*

(1) $\mathrm{Formula}(Z, i, j)$*, meaning* $Z[i, j]$ *is a formula,*

(2) $\mathrm{QBF}(Z, i, j)$*, meaning* $Z[i, j]$ *is a* QBF*, and*

(3) $\mathrm{Sequent}(Z, i, j)$*, meaning* $Z[i, j]$ *is a sequent.*

Proof sketch. (1) The string $Z[i, j]$ is a formula, if

1. it has an equal number of "(" and ")",
2. for each $u \in \{i, \ldots, j-1\}$ the number of opening parentheses "(" in $Z[i,u]$ greater than the number of closing ones ")", and
3. every string of length 2, which does not occur in a well bracketed formula, is not allowed to be a substring of $Z[i,j]$. Examples of length 2 strings, which are not allowed to occur in a formula: px, $\neg\exists$.

Each of these three statements is FOM-expressible, because, as mentioned before, counting is possible in $\mathbf{TC^0}$.

(2) $Z[i,j]$ is a QBF, if

1. it is a formula, which is FOM expressible as shown in (1), and
2. every occurrence of an x-variable in $Z[i,j]$ should be in the scope of some quantifier (x should be in a subformula of $Z[i,j]$ whose outer connective is either $\forall$ or $\exists$), this fact can be also checked by counting parentheses.

(3) This case follows from (1) and (2) since a sequent is a sequence of formulas separated by commas. □

Note that parentheses play a very important role in parsing formulas, which is why they need to follow a strict syntax, in which parentheses are not optional. They will always be used to mark the scope of each connective or quantifier, for example $(A \wedge B) \vee C$ should be $((A \wedge B) \vee C)$ and $\forall x \exists y A(x,y)$ should be $(\forall x(\exists y A(x,y)))$.

Lemma 6.1.4. *There is a* $\mathbf{TC^0}$ *function that outputs* $\mathrm{ID}_\varphi(\psi)$ *given* Z, i, j, a, b *such that* $i \leq a \leq b \leq j$, *and* $Z[i,j], Z[a,b]$ *encode the formulas* φ, ψ *respectively.*

Proof. Let $f : x \mapsto \mathrm{ID}_\varphi(\psi)$, with $x = (Z, i, j, a, b)$, be the formula described in the lemma. Let $R_f(x,k,b)$ be a predicate representing the statement "the kth bit of $\mathrm{ID}_\varphi(\psi)$ is b". To prove that $\mathrm{ID}_\varphi(\psi)$ is $\mathbf{TC}^0$ recognizable, it is enough to show that the predicate $R_f(x,k,b)$ is in $\mathbf{TC}^0$.

This is indeed the case, since we can describe how this function works as follows.

By the definition of $\mathrm{ID}_\varphi(\psi)$, b is either the empty string, 1, or 2.

b is not the empty string (1 or 2) if and only if

(i) there exists l, m such that $Z[l,m]$ is a subformula of φ,

(ii) ψ is a subformula of $Z[l,m]$, and

(iii) the number of "(" in $Z[i,l-1]$ minus the number of ")" in $Z[i,l-1]$ is equal to i. This intuitively means a bracket is still not closed before the subformula, which entails that ψ should be in the scope of a logical connective.

Thus if these conditions are satisfied and $\alpha_{l-1} \in \{\wedge, \vee\}$ then b is 2. If the conditions are satisfied and α_{l-1} is neither $\wedge$ nor $\vee$ then b is 1, otherwise b is the empty string. $\square$

Lemma 6.1.5. *Let*

- ExistsLeft(S_i, S_j),
- ExsitsRight(S_i, S_j),
- ForAllLeft(S_i, S_j), *and*
- ForAllRight(S_i, S_j)

be predicates, such that each is true if and only if sequent S_i can be derived from sequent S_j by the corresponding quantifier rule.

These predicates are FOM-expressible.

Proof. The goal is to describe, for each of these predicates, an FOM formula which expresses it.

In what follow we describe the formula $\varphi_{\mathrm{ER}} = \varphi_1 \wedge \varphi_2$ which expresses the predicate ExistsRight.

Let φ_1 be the formula expressing that the sequents S_i and S_j are identical except for the principal formula and the auxiliary formula, denoted by P and A respectively, and that the principal formula P is of the form $\exists x\ C$ for some variable x.

Let φ_2 express that there exists a propositional subformula B of A and a variable x such that A is the result of substituting every occurrence of x in C by B, (i.e, $P = \exists x\ C$ and $A = C[x/B]$), which can be described as follows: For each subformula C' of C, there exist a subformula A' of A where $\mathrm{ID}_C(C') = \mathrm{ID}_A(A')$ such that

- if C' is the atomic formula x then $A' = B$,
- if C' is atomic but not x then $C' = A'$, and
- if C' is not atomic then C' and A' have the same principal (top-level) connective.

All of the mentioned conditions are indeed FOM expressible.

A formula φ_{EL} expressing the predicate ExistsLeft is constructed similarly but with an additional condition which ensure that B is the atomic formula x_a such that x_a is a free variable (an eigenvariable) that does not occur in S_i (the lower sequent).

The formulas φ_{FL} and φ_{FR} expressing the predicates ForAllLeft and ForAllRight can be described analogously. $\square$

Definition 6.1.3
Define $\text{Inf}_1(S_i, S_j)$ to be true if and only if sequent S_i is derivable from sequent S_j by a unary inference rule, where S_i, S_j are given as strings over QBFs. Similarly, $\text{Inf}_2(S_i, S_j, S_k)$ is true if and only if S_i is derivable from two sequents S_j and S_k by a binary inference rule. $\triangleleft$

Lemma 6.1.6. *Both* Inf_1 *and* Inf_2 *are FOM-expressible.*

Proof.

- We first consider $\text{Inf}_1(S_i, S_j)$.

  The previous lemma states that there exists an FOM formula that expresses that S_i is derived from S_j by applying a quantifier rule.

  The structural rules as well as the unary propositional rules are easily expressible by an FOM formula, for example, for the rule weakening right we just need to check that S_i and S_j are identical except for the rightmost formula in S_i.

- Next we consider $\text{Inf}_2(S_i, S_j, S_k)$.

  We have four binary inference rules:

  ◦ For the rule $\wedge$r, the FOM formula should check:
    (i) S_i, S_j and S_k are identical except for their rightmost formula. Meaning if $Z_i[1, l_i]$ is the string representing S_i, $Z_j[1, l_j]$ representing S_j, and $Z_k[1, l_k]$ representing S_k, then there is some $\alpha \leq l_i, l_j, l_k$ such that $Z_i[1, \alpha] = Z_j[1, \alpha] = Z_k[1, \alpha]$ and $Z_i[\alpha, l_i], Z_j[\alpha, l_j]$, and $Z_k[\alpha, l_k]$ are formulas.
    (ii) If A is the rightmost formula of S_j and B is the rightmost formula of S_k then $A \wedge B$ is the rightmost formula of S_i

  ◦ For the rule $\vee$l, analogously the formula should check:
    (i) S_i, S_j and S_k are identical except for their leftmost formula.
    (ii) If A is the leftmost formula of S_j and B is the leftmost formula of S_k then $A \vee B$ is the leftmost formula of S_i

  ◦ For the rule $\supset$l, the formula should check:
    (i) If the rightmost formula in S_j is φ and the leftmost formula in S_k is ψ, then the rightmost formula in S_i is $\varphi \supset \psi$, and
    (ii) S_i without the leftmost formula is identical to S_j without the rightmost formula, and S_k without the leftmost formula.

  ◦ For the cut rule, the FOM formula should check:
    (i) S_i, S_j without the rightmost formula and S_k without the leftmost formula are identical.
    (ii) The rightmost formula of S_j and the leftmost formula of S_k are identical

□

Lemma 6.1.7. *The following predicates are FOM-expressible:*

- $\text{Proof}_{G_0}(Z,i,j)$*, which is true if and only if* $Z[i,j]$ *is a* G_0*-proof,*
- $\text{Prototype}(Z,i,j,u,v)$*, which holds if and only if* $Z[i,j]$ *is a* G_0*-proof of a QBF in prenex form and* $Z[u,v]$ *is a* π*-prototype, and*
- $\text{Component}(Z,i,j,k,l)$*, which is true if and only if* $Z[i,j]$ *is the* k*th component of the* l*th* π*-prototype* A_l*.*

Proof.

- $\text{Proof}_{G_0}(Z,i,j)$

  From Lemma 6.1.3, we know that recognizing sequents is FOM-expressible, and by Lemmas 6.1.5 and 6.1.6 checking if a sequent is derived by applying an inference or a quantifier rule on previous sequents is also FOM-expressible.

  This implies that we can use the predicates defined in these lemmas to define the predicate $\text{Proof}_{G_0}(Z,i,j)$ thus it is FOM-expressible.

- $\text{Prototype}(Z,i,j,u,v)$

  Since Proof_{G_0} is FOM-expressible as well as the quantifier introduction rules which means we can express with an FOM formula that $Z[i,j]$ is a G_0-proof and that $Z[u,v]$ is the auxiliary formula of a quantifier introduction step.

- $\text{Component}(Z,i,j,k,l)$

  This also can be expressed by an FOM formula, since we can identify prototypes with the predicate Prototype.

  Similarly to φ_2 in the proof of ExistsRight in Lemma 6.1.5, we can express that there exists a subformula B of the prototype A and a variable x such that A is the result of substituting every occurrence of x in C by B, (C being the propositional part of the end-sequent of the proof, thus B would be a component of the prototype). □

Theorem 6.1.8. *The* Σ_1^q*-Witnessing problem for* G_0 *is solvable by an* $\mathbf{NC}^1$*-function*

Proof. As proved by Buss a problem is solvable by an $\mathbf{NC^1}$-function if and only if there exists an **Alogtime** Turing machine that solves it. This means it is enough to describe an **Alogtime** Turing machine that can recognize the π-witness formulas, given a string Z which encodes a G_0-proof and an assignment to the free variables of a Σ_1^q-formula.

In the previous lemmas we proved that whether a string over Σ_{QBF} is a formula, π-prototype, a component of a prototype, or a G_0-proof is $\mathbf{TC^0}$-recognizable and thus

Alogtime-recognizable. If we fix an ordering on the π-prototypes to be the order in which they appear in Z. We can now define the **Alogtime** Turing machine M as follows.

M takes as input the tuple (Z, i, c), such that $|Z| = n$, and accepts it if and only if there exists $r \in \{1, \ldots, n\}$ such that

- $Z[1, r]$ encodes π, a G_0-proof of a QBF,
- $Z[r+1, n]$ encodes a truth assignment $\overline{v}$, and
- η_i (the ith π-witness formula) evaluates to $c \in \{\top, \bot\}$ under $\overline{v}$.

The first two involve the $\mathbf{TC^0}$ predicates mentioned earlier, and since the evaluation of a Boolean formula under a given assignment is in **Alogtime**, M can guess a j and then verify that φ_i^j, φ_j, and $\neg\varphi_l$ for each $l < j$ are true under $\overline{v}$ (such that φ_j and φ_l are prototypes, and φ_i^j is the ith component of the prototype φ_j).

In each computation path, M needs to guess indices of Z, each of which has $\lceil log\ n \rceil$ bits, which entails that M runs in alternating time $O(log\ n)$. $\square$

6.2 The Σ_1^q-Witnessing problem for G_0^*

Recall that G_0^* is the system for treelike proofs for QBFs, in which the cut rule is restricted to quantifier-free (propositional) formulas only.

Definition 6.2.1
Let F and G be two functions. We say that F is many-one $\mathbf{AC^0}$-reducible to G if there exist two $\mathbf{AC^0}$-function g, h such that $F(x) = g(G(h(x)))$. $\triangleleft$

Definition 6.2.2
A function F is said to be hard for $\mathbf{FNC^1}$ under many-one $\mathbf{AC^0}$-reductions if and only if every $\mathbf{NC^1}$-function is many-one $\mathbf{AC^0}$-reducible to it. F is complete for $\mathbf{FNC^1}$ if F itself is in $\mathbf{FNC^1}$. $\triangleleft$

Theorem 6.2.1. Witness$[G_0^*, \Sigma_1^q]$ *is hard for* $\mathbf{FNC^1}$ *under many-one* $\mathbf{AC^0}$*-reductions.*

Proof.

Let f be a **DLogtime**-uniform $\mathbf{NC^1}$-function. The goal is to find an $\mathbf{AC^0}$-reduction to Witness$[G_0^*, \Sigma_1^q]$.

We assume without loss of generality that there exists a polynomial $p(\cdot)$ such that

$$\text{for any } n \qquad f : \{0,1\}^n \longrightarrow \{0,1\}^{p(n)}.$$

Since every $\mathbf{NC^1}$ predicate is computed by a **DLogtime**-uniform family of polynomial-size propositional formulas, and since the predicate $R_f(x, i, c)$ (which denotes that the ith bit of $f(x)$ is c) is in $\mathbf{NC^1}$, there exists a **DLogtime**-uniform family of polynomial-size propositional formulas $\{A_n\}_n$, such that for each $n = |x|$, $A_n(x, i)$ evaluates to true if and only if the ith bit of $f(x)$ is 1, where x and i are represented in A_n by the propositional variables $\overline{q}$ and $\overline{r}$ respectively.

Let $m = p(n)$, and for each $i \in \{1, \dots, m\}$ let $\overline{i}$ represent the truth assignments of the variables in $\overline{r}$. We define the sequent S_n as follows :

$$\longrightarrow \exists y_1 \cdots y_m[(y_1 \leftrightarrow A_n(\overline{q}, \overline{1})) \wedge \ldots \wedge (y_m \leftrightarrow A_n(\overline{q}, \overline{m}))]$$

where $y_i \leftrightarrow A_n(\overline{q}, \overline{i})$ is used as an abbreviation of $(y_i \wedge A_n(\overline{q}, \overline{i})) \vee (\neg y_i \wedge \neg A_n(\overline{q}, \overline{i}))$.

Assume π_n is the G_0^*-proof of S_n, and $\overline{v}$ is the truth assignments to the variables in $\overline{q}$. Then, given the solution for Witness$[G_0^*, \Sigma_1^q]$ over $(\pi_n, \overline{v})$, i.e., an assignment $\overline{u}$ to the $\overline{y}$ variables, we can get an $\mathbf{AC^0}$-function that computes f (since by the definition of S_n, y_i is true if and only if $A_n(\overline{q}, \overline{i})$).

It remains to show that an $\mathbf{AC^0}$-function can output for each x a G_0^*-proof $\pi_{|x|}$ for $S_{|x|}$ of size polynomial in $|x|$. We outline in what follows such a proof.

For each $i \in \{1, \dots, m\}$, we can construct a G_0^*-proof for the sequent

$$\longrightarrow A_{|x|}(\overline{q}, \overline{i}) \leftrightarrow A_{|x|}(\overline{q}, \overline{i}),$$

by repeatedly applying $\wedge$r on these sequents, we get

$$\longrightarrow (A_{|x|}(\overline{q}, \overline{1}) \leftrightarrow A_{|x|}(\overline{q}, \overline{1})) \wedge \ldots \wedge (A_{|x|}(\overline{q}, \overline{m}) \leftrightarrow A_{|x|}(\overline{q}, \overline{m})).$$

After m application of $\exists$r we get the proof $\pi_{|x|}$ of

$$\longrightarrow \exists y_1 \cdots y_m[(y_1 \leftrightarrow A_{|x|}(\overline{q}, \overline{1})) \wedge \ldots \wedge (y_m \leftrightarrow A_{|x|}(\overline{q}, \overline{m}))].$$

An $\mathbf{AC^0}$-function can output π_n because each line of π_n has a highly uniform structure and it is easy to determine what the jth sequent of π_n should look like for any j. $\square$

We conclude then the next theorem from theorem 6.1.8 and theorem 6.2.1.

Theorem 6.2.2. *Witness$[G_0, \Sigma_1^q]$ and Witness$[G_0^*, \Sigma_1^q]$ are both complete for* $\mathbf{FNC^1}$ *under many-one* $\mathbf{AC^0}$ *reduction.*

CHAPTER 7

Summary

Proving whether a given QBF evaluates to true is a **PSPACE**-complete problem [SM73]. Many problems from application domains like model checking or formal verification are known to be **PSPACE**-complete, hence QBFs provide us with a powerful tool to encode them. Proving lower bounds with the help of a solid proof system for QBFs can help us extract strategies to improve QBF solvers, which is why the work of researchers like Morioka, Cook, Krajíček, Pudlák, and others is extremely important.
The literature in this research area is often very hard to read, because proofs are often sketched and important details are omitted in intricate proof constructions. A witness for this fact is [KP90] or [Kra95]. We aim with this thesis to make the work of these scholars more readable and accessible.

To get the full picture, complexity classes that are relevant to the results provided by Cook and Morioka are first presented. Classes from the polynomial hierarchy are interesting here because QBFs in prenex normal form define a hierarchy of formulas Σ^q_∞ that corresponds to the polynomial hierarchy (**PH**) [Wra76]. Hence any problem from the **PH** can be represented by a prenex QBF with a specific quantifier prefix.

Circuits complexity classes on the other hand, are the main ingredient used in proving the complexity of the Σ^q_1-Witnessing problem for G_0 and G^*_0, and that is based on a first-order characterization of the classes $\mathbf{AC^0}$ and $\mathbf{TC^0}$.

The last section in the complexity chapter was dedicated to the complexity classes of local search problems. We presented in this thesis the classes **FP**, **FNP**, and **PLS**, and their relation to the polynomial hierarchy to better understand the results proved in [CM05] and [Mor05] for the general witnessing problem.

Krajíček and Pudlák introduced in [KP90] the sequent calculi KPG for QBFs, and defined a hierarchy of fragments KPG$_i$, KPG$^*_i$, such that a KPG$_i$-proof is restricted to $\Sigma^q_i \cup \Pi^q_i$ formulas, and KPG$^*_i$ is KPG$_i$ but restricted to treelike proofs. Those systems are closely related to the theories S^i_2 and T^i_2 of bounded arithmetic [Bus86]. By modifying the definition of those systems, Cook and Morioka in [Mor05, CM05] introduced new systems which have a better and more natural correspondence to bounded arithmetic. They introduced two restrictions. In their system G_i any QBF can be handled, thus, in contrast to the corresponding system of Krajíček and Pudlák, the new system is complete. Furthermore, Cook and Morioka restricted the application of the cut rule to $\Sigma_i^q \cup \Pi_i^q$ formulas, and restricted the target formula in the quantifier rules to be quantifier-free. With the new definition they introduced two new complete proof systems for proving QBFs, G_0 and G_0^*, in which the cut rule is restricted to quantifier-free (i.e., propositional) formulas. Cook and Morioka also proved that G and KPG are p-equivalent. In this thesis we presented a more detailed version of the p-equivalence proof in Lemma 3.2.2 (Lemma 5.10. [Mor05], Lemma 1. [CM05])), and Lemma 3.2.1 (Lemma 5.8. in [Mor05], Lemma 2. in [CM05]).

One of the very important and trickiest result presented in this thesis is the strong connection between QBFs and fragments of bounded arithmetic, however the hardest part is the additional detailed description of the relation between the system G (and its restrictions) and restricted version of the theories S_2 and T_2 in bounded arithmetic.

Theorem 4.5.6 (Theorem 9.2.5 in [Kra95]) describes how a bounded arithmetic proof of a formula A can be translated to a QBF proof of its translation $||A||$. Based on the proof outlined by Krajícek a more detailed proof is presented in this work. However some sequents used in the proof are not proven in detail, but rather taken from [Kra95] without proof. As an effort to complete the proof, Corollary 4.5.1.1 and Lemma 4.5.2 were introduced. To make the proof more readable two parts of the proof are introduced as lemmas before the theorem, namely Lemma 4.5.4 and Lemma 4.5.5.

Because of the connection between QBFs and bounded arithmetic, Cook and Morioka were able to use results proven in bounded arithmetic to prove that the Σ_1^q-Witnessing problem for G_1^* is **FP**-complete and the Σ_1^q-Witnessing problem for G_1 is **PLS**-complete (Theorem 7 in [CM05], Theorem 6.2 in [Mor05]). Though these proofs are not presented in this thesis, the theorem itself, Theorem 6.0.1, is included because one of its implications is that G_1^* does not necessarily p-simulate G_1. This entails that the fact that G_0^* p-simulate G_0 for proving Σ_1^q-formulas is not a trivial result.

First-order theories T axiomatized by purely universal formulas, have an LK-proof with all cuts on quantifier-free formulas as proved by Buss in [Bus98], which seem similar to the two complete proof systems G_0 and G_0^*, first introduced by Cook and Morioka in [CM05] and by Morioka in [Mor05]. This fact motivated the study of first-order results for QBFs, results which were proved by Buss for the theories T in [Bus98]. This lead to a polynomial-time version of the midsequent theorem for G_0^*, i.e., Theorem 5.2.2, and a version of Herbrand Theorem for G_0, i.e., Theorem 5.1.1. These results were the base that was used to describe an $\mathbf{NC^1}$ algorithm for Σ_1^q-Witnessing problem for G_0 and G_0^*.

Though Cook and Morioka provided complexity results for Σ_i^q-Witnessing for G_i and for G_i^*, Σ_1^q-Witnessing for G_1 and for G_1^*, and Σ_1^q-Witnessing for G_0 and G_0^*, the complexity of Σ_i^q-Witnessing problem for G_0 and G_0^* for $i > 1$ is still open.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[BCE$^+$98] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3 – 19, 1998.

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

[Bus87] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *STOC*, 1987.

[Bus88] Samuel R. Buss. Weak formal systems and connections to computational complexity. Available at `https://www.math.ucsd.edu/~sbuss/ResearchWeb/weakformaltopics/`, January-May 1988. Student-written lecture notes at U.C.Berkley.

[Bus91] Samuel R. Buss. Propositional consistency proofs. *Annals of Pure and Applied Logic*, 52(1):3 – 29, 1991.

[Bus98] Samuel R. Buss. Chapter 1: An introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*. Elsevier, 1998.

[CM05] Stephen A. Cook and Tsuyoshi Morioka. Quantified propositional calculus and a second-order theory for NC^1. *Archive for Mathematical Logic*, 44:711–749, 2005.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, page 83–97, New York, NY, USA, 1975. Association for Computing Machinery.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44:36–50, 1979.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210 und 405–431, 1935.

[Gen69] 3. investigations into logical deduction. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, volume 55 of *Studies in Logic and the Foundations of Mathematics*, pages 68 – 131. Elsevier, 1969.

[Imm99] Neil Immerman. Descriptive complexity. In *Graduate Texts in Computer Science*, 1999.

[JPY85] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 39–42, 1985.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KP90] Jan Krajícek and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Math. Log. Q.*, 36:29–46, 1990.

[Kra95] Jan Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Cambridge University Press, 1995.

[Kre88] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490 – 509, 1988.

[MBIS90] David A. Mix-Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, December 1990.

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

[Pap94] Christos H. Papadimitrio. *Computational complexity.* Addison-Wesley, 1994.

[Ruz79] Walter L. Ruzzo. On uniform circuit complexity. *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 312–318, 1979.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time. In *STOC 1973*, 1973.

[Tak87] Gaisi Takeuti. *Proof theory*. North-Holland, 2nd edition, 1987.

[Wra76] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976.