Optimal proof systems and uniform systems.

Razafindrakoto, Jean-Jose

Optimal Proof Systems and Uniform Systems

Jean-José Razafindrakoto

February 7, 2012

A thesis submitted to Swansea University in
candidature for the degree for the Degree of Master of Research

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

<date> Wednesday 8 th 2012
Signed:

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of an MRes in Logic and Computation.

<date> Wednesday 8th 2012
Signed:

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

<date> Wednesday 8th 2012
Signed:

Statement 3

I hereby give my consent for my thesis to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

<date> Wednesday 8th 2012
Signed:

Abstract

In "Uniform Proof Complexity", Beckmann introduced the notion of the Uniform Reduct of a proof system which he defined to be the set of those true bounded formulae (in the language of Peano Arithmetic) which have polynomial-size proofs under the Paris-Wilkie translation . In his comments to Beckmann's paper, Cook pointed out that the existence of a proof system whose uniform reduct is the set of all true Σ_0^B-formulae is equivalent to the existence of an optimal proof system. In this work, we carry out a detailed proof of that equivalence.

Contents

1 Introduction

The P vs NP problem is arguably the most important problem in both Computer Science and Mathematics. As a matter of fact, it is the first of seven million-dollar Millenium Prize Problems listed by the Clay Mathematics Institute [Coo03]. Besides, NP-completeness is debatably the most pervasive concept in Computer Science since it captures the computational complexity of many significant problems from different areas of the field (see [GJ90] for many examples).

One major way towards a solution to the P vs NP problem is Propositional Proof Complexity, an area of study developed by Cook and Reckow in their seminal paper entitled "The Relative Efficiency of Propositional Proof Systems" [CR79], where they showed that NP = coNP if and only if a polynomially bounded proof system exists (a polynomially bounded propositional proof system, roughly speaking, is a polynomial-time proof-verifier P for membership in TAUT, the set of all propositional tautologies, such that every tautology has a polynomial-size proof in P in the length of the tautology). In Propositional Proof Complexity, the basic task is to prove that stronger and stronger proof systems are not polynomially bounded, until it is established for all proof systems. Hence, if one achieves that general program of Propositional Proof Complexity described above, then NP is different to coNP, thus separating P from NP.

In keeping with the general program of Propositional Proof Complexity, a lot of work has been done in deriving strong lower bounds for various standard propositional proof systems. For example, it has been shown by Haken, in [Hak85], that the Pigeonhole Principle requires exponential size Resolution refutations. Later, Beame and Pitassi provided an improved lower bound on the the sizes of Resolution refutations for the Pigeonhole Principle in [BP96]. However, unlike Resolution (and other propositional proof systems like AC^0-Frege systems and their extensions), no strong lower bounds are known for Frege and Extended Frege systems. The best lower bounds known for them are linear on the number of lines and quadratic on the number of symbols [Bus02] (the Pigeonhole Principle requires polynomial length Frege [Bus87b] and Extended Frege [CR79] proofs).

Since some families of tautologies require polynomial size proofs in some propositional proof systems and exponential size proofs in others, one can then think of comparing propositional proof systems according to their relative efficiencies. To do that, Cook and Reckow defined, in [CR79], the notion of p-simulation. Informally, a propositional proof system P_1 p-simulates another propositional proof system P_2, means that there exists a polynomial time procedure that translates every proof in P_2 into a proof in P_1. A weaker notion of p-simulation between two propositional proof systems, called simulation, also exists, where the existence of a polynomial time procedure is not required. Given these informal definitions, one important question arises: Is there any propositional proof system which simulates every other propositional proof system? In other words, does there exist an optimal proof system?

If an optimal proof system exists, then in order to separate NP from coNP it would suffice to prove that such a system is not polynomially bounded. Partial results have been obtained in [KP89a, MT98, BdG98], relating the existence of optimal proof systems to the equivalence of certain complexity classes. More recently, Cook pointed out in [Coo06] that the existence of an optimal proof system is equivalent to the existence of a propositional proof system such that its uniform reduct equals the set of all true Σ_0^B-formulae. Here, the uniform reduct of a propositional proof system (or just uniform system) is a notion defined by Beckmann in [Bec05] and is the set of those true Σ_0^B-formulae which have polynomial size proofs under some translation in the style of the Paris-Wilkie translation.

The goal of this project is to carry out the detailed proof of the equivalence between the existence of an optimal proof system and the existence of a propositional proof system whose uniform reduct equals the set of all true Σ_0^B-formulae.

In Section 2, we introduce some basic background of complexity theory that is needed for our purpose. Then, Section 3 gives an overview of Propositional Proof Complexity and the definitions that we need for later sections. In Section 4, we define Σ_0^B-formulae and show how to translate them into propositional logic. From there, we formally define the Uniform Reduct of a propositional proof system. Finally, Section 5 is the main body of this dissertation. In there, we show how to encode Polish propositional formulae, truth assignments, polytime Turing machine computations and the reflection principle for a propositional proof system. Additionally, we present the detailed-proof of the equivalence between the existence of an optimal proof system and the existence of a propositional proof system whose uniform reduct equals the set of all true Σ_0^B-formulae.

2 Complexity Theory

In this section, we first introduce our model of computation, which is a Turing machine. From there, we define what it means for a language to be in the complexity class NP. Then, we define the notion of NP-completeness.

2.1 Turing Machines

Our exposition of Turing machines (deterministic and non-deterministic Turing machines) follows [Pap94].

Notation We use Σ^* (resp. Σ^+) to denote the set of all finite (resp. non-empty finite) strings over the finite alphabet Σ under consideration. Additionally, $\mathbb{N}$ denotes the set of natural numbers including 0 and $\mathbb{Z}$ denotes the set of all integers.

2.1.1 Deterministic Turing Machines

The deterministic Turing machine that we are going to describe consists of a string of symbols from a finite alphabet, a finite state control and a cursor that scans the symbols on the string and that is connected to the control. Depending on the state of the control and the symbol scanned by the cursor, the machine assumes a new state, overwrites the symbol scanned by the cursor and moves the cursor to the left or right of the overwritten symbol, or just leaves the cursor at its current position.

Definition 2.1 *Define a **deterministic Turing machine** M to be a quadruple (K, Σ, δ, s) where*

1. *K is a finite set of states.*

2. *Σ is a finite set of symbols and is called the **alphabet** of M. Assume that $K \cap \Sigma = \emptyset$. Furthermore, assume that Σ always contains the symbols $\sqcup$ **(blank symbol)** and $\triangleright$ **(first symbol)**.*

3. *δ is a transition function from $K \times \Sigma$ to $(K \cup \{h, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$, where h is the **halting state**, yes is the **accepting state**, no is the **rejecting state**, $\leftarrow$ is the cursor direction for left, $\rightarrow$ is the cursor direction for right, $-$ is the cursor direction for stay. Assume that $(\{h, yes, no\} \cup \{\leftarrow, \rightarrow, -\}) \cap (K \cup \Sigma) = \emptyset$.*

4. *$s \in K$ is the **initial state**.*

Notation . When we write Γ (possibly subscripted), we always mean $\Sigma \backslash \{\sqcup\}$ and $\Gamma^* \subseteq (\Sigma \backslash \{\sqcup\})^*$, for some Turing machine's alphabet Σ.

For every $q \in K$ and $\sigma \in \Sigma$, there exists a $q' \in K \cup \{h, yes, no\}$, a $\sigma' \in \Sigma$ and a $D \in \{\leftarrow, \rightarrow, -\}$ such that $\delta(q, \sigma) = (q', \sigma', D)$, where q is the current state of the control, σ is the symbol scanned by the cursor, q' is the new state, σ' is the symbol to be overwritten on σ and D is the the direction in which the cursor will move. Assume that if $\sigma = \triangleright$, then $\sigma' = \sigma$ and $D = \rightarrow$.

M works as follows. Initially, the initial state is s; the string is initialised to $\triangleright x$, where $x \in \Gamma^*$ and x is called the **input** of M; the cursor scans $\triangleright$. Then M moves according to the transition function δ. Now, M halts if one of the following states is reached: h, yes or no. If the yes state is reached, then M **accepts** its input; if the no state is reached, then M **rejects** its output. If M halts on input x, then define the **output** of M on x, denoted $M(x)$, as follows. If M accepts x, then $M(x) = yes$; if M rejects x, then $M(x) = no$; if the state h is reached, then the string, at the time of halting, consists of $\triangleright y$ (y is a finite string whose last symbol is different from $\sqcup$), possibly followed by a string of blanks, and we consider that $M(x) = y$.

Definition 2.2 *Let M be a Turing machine. A **configuration** of M is a triple (q, w, u) where $q \in K \cup \{h, yes, no\}$, $w \in \Sigma^+$, and $u \in \Sigma^*$. w is the string to the left of the cursor such that the last symbol of w is the current symbol scanned by the cursor. u is the string (may be an empty string) to the right of the cursor. Finally, q is the current state.*

Definition 2.3 *Let M be a Turing machine and $w = v\sigma$, where $v \in \Sigma^*$, and $\sigma \in \Sigma$. Configuration (q, w, u) **yields** configuration (q', w', u') **in one step**, denoted $(q, w, u) \xrightarrow{M^1} (q', w', u')$, if in the transition function, $\delta(q, \sigma) = (q', \sigma', D)$ and: if $D = -$, then $w' = v\sigma'$ and $u' = u$; if $D = \rightarrow$, then w' is $v\sigma'$ with the first symbol of u appended to it ($\sqcup$ if u is the empty string) and u' is u with its first symbol omitted (if u is the empty string, then u' remains empty); if $D = \leftarrow$, then $w' = v$ and u' is u with σ' attached in the beginning.*

*Define the notion of configuration (q, w, u) **yields** configuration (q', w', u') **in k steps**, denoted $(q, w, u) \xrightarrow{M^k} (q', w', u')$ and where $k \geq 0$, as follows.*

$(q, w, u) \xrightarrow{M^k} (q', w', u')$ *for* $k \geq 0$, *if there are configurations* (q_j, w_j, u_j), *for* $j = 1, \ldots, k+1$, *such that* $(q_i, w_i, u_i) \xrightarrow{M^1} (q_{i+1}, w_{i+1}, u_{i+1})$, *for* $i = 1, \ldots, k$, *and* $(q_1, w_1, u_1) = (q, w, u)$ *and* $(q_{k+1}, w_{k+1}, u_{k+1}) = (q', w', u')$.

At last, (q, w, u) ***yields*** (q', w', u') ***in at least one step****, denoted* $(q, w, u) \xrightarrow{M^+} (q', w', u')$, *if there exists a* $k \geq 1$ *such that* $(q, w, u) \xrightarrow{M^k} (q', w', u')$.

Definition 2.4 *Given a Turing machine* M *and a language* $L \subseteq \Gamma^*$, *we say that* M ***decides*** L *iff for every* $x \in \Gamma^*$ *the following conditions hold: if* $x \in L$, *then* $(s, \triangleright, x) \xrightarrow{M^+} (yes, w, u)$, *and, if* $x \notin L$, *then* $(s, \triangleright, x) \xrightarrow{M^+} (no, w, u)$ *for some* w, u.

Definition 2.5 *Let* M *be a Turing machine and* L *be a language such that* $L \subseteq \Gamma^*$. M ***decides*** L ***in time*** $f(n)$ *iff the following two conditions hold:* M *decides* L; *for any* $x \in \Gamma^*$, *if* $(s, \triangleright, x) \xrightarrow{M^k} (H, w, u)$, *for* $H \in \{yes, no\}$, *then* $k \leq f(|x|)$.

Definition 2.6 *Let* f *be a function from* $(\Sigma \backslash \{\sqcup\})^*$ *to* Σ^*. *Then* f *is said to be* ***computable*** *if and only if there exists a deterministic Turing machine* M *with alphabet* Σ *such that for all* $x \in (\Sigma \backslash \{\sqcup\})^*$, $M(x) = f(x)$. f *is said to be* ***computable in time*** $g(n)$ *if and only if* M *is computable and for all* $x \in (\Sigma \backslash \{\sqcup\})^*$, $(s, \triangleright, x) \overset{M^k}{\to} (h, w, u)$ *and* $k \leq g(|x|)$.

2.1.2 Non-deterministic Turing Machines

The definition of a **non-deterministic Turing machine** is much like the deterministic Turing machine one, except that δ is no longer a transition function but now a relation Δ such that $\Delta \subset (K \times \Sigma) \times [(K \cup \{h, yes, no\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}]$.

The definition of a non-deterministic Turing machine's **configuration** is exactly the same as the definition of a deterministic Turing machine's configuration. However, a non-deterministic Turing machine's configuration may now yield more than one configuration in one step.

Definition 2.7 *Let* N *be a nondeterministic Turing machine and* $w = v\sigma$, *where* $v \in \Sigma^*$ *and* $\sigma \in \Sigma$. *Configuration* (q, w, u) ***yields*** *configuration* (q', w', u') ***in one step****, denoted* $(q, w, u) \xrightarrow{N^1} (q', w', u')$, *if there exists a rule* $((q, \sigma), (q', \sigma', D))$ *in* Δ *such that: if* $D = -$, *then* $w' = v\sigma'$ *and* $u' = u$; *if* $D = \rightarrow$, *then* w' *is* $v\sigma'$ *with the first symbol of* u *appended to it* ($\sqcup$ *if* u *is the empty string) and* u' *is* u *with its first symbol omitted (if* u *is the empty string, then* u' *remains empty); if* $D = \leftarrow$, *then* $w' = v$ *and* u' *is* u *with* σ' *attached in the beginning.*

$\xrightarrow{N^k}$ *can be defined in the same way as* $\xrightarrow{M^k}$. *Finally,* $(q, w, u) \xrightarrow{N^+} (q', w', u')$ *if there exists a* $k \geq 1$ *such that* $(q, w, u) \xrightarrow{N^k} (q', w', u')$.

Definition 2.8 *Let* N *be a non-deterministic Turing machine and* L *be a language such that* $L \subseteq \Gamma^*$. N ***decides*** L ***in time*** $f(n)$ *iff for any* $x \in \Gamma^*$ *the following two conditions hold:*

- *for every configuration* C *that arises in the computations of* N *on* x, *there exists a* $k \in \mathbb{N}$ *such that* $k \leq f(|x|)$ *and* $(s, \triangleright, x) \xrightarrow{N^k} C$;

– $x \in L$ *iff* $(s, \triangleright, x) \xrightarrow{N^+} (yes, w, u)$.

2.2 NP and NP-completeness

Definition 2.9 *A language L belongs to* NP *iff there exists a non-deterministic Turing machine N and a polynomial p such that N decides L in time $p(n)$.*

Theorem 2.10 *A language L belongs to* NP *if and only if there exists a polynomial time Turing machine V (called a proof verifier) and a polynomial p such that for all $x \in \Gamma^*$, the following holds:*

(1) $$x \in L \Leftrightarrow \exists \pi \in \Gamma^*(|\pi| \leq p(|x|) \wedge V \text{ accepts } (x, \pi))$$

Proof: Before we start the proof, it is worth pointing out that we view configurations and sequences of configurations as strings over a finite alphabet Γ which includes the symbols "(", ")" and ",". Obviously, the length of a string over Γ corresponds to the number of symbols in the string.

($\Rightarrow$) Suppose that $L \in$ NP. By Definition 2.9, we can let N be a non-deterministic Turing machine and p_1 be a polynomial such that N decides L in time $p_1(n)$. Define V to be a polynomial time Turing machine that takes as its input pairs of strings (x, π), where $x, \pi \in \Gamma^*$, and checks:

1. if $\pi = C_1, \ldots, C_j$;
2. if $C_1 = (s, \triangleright, x)$;
3. if $C_j = (yes, w, u)$ for some w, u;
4. if $C_i \xrightarrow{N^1} C_{i+1}$ for all $i = 1, \ldots, j-1$.

If all these four conditions are satisfied, then V accepts (x, π). Let x be an arbitrary string in Γ^*. Now, prove (1) and define p_2 in the course of the proof.

Suppose that $x \in L$. Show that $\exists \pi(|\pi| \leq p_2(|x|) \wedge V$ accepts $(x, \pi))$ holds. Since N decides L in time $p_1(n)$, by Definition 2.8, we have $(s, \triangleright, x) \xrightarrow{N^{j-1}} (yes, w, u)$ where $2 \leq j \leq p_1(|x|) + 1$. So, let $S = C_1, \ldots, C_j$ such that $C_1 = (s, \triangleright, x)$, $C_j = (yes, w, u)$, and $C_i \xrightarrow{N^1} C_{i+1}$ for all $i = 1, \ldots, j-1$. Let $\pi = S$. Clearly, V accepts (x, π). Now, derive an upperbound for $|\pi|$. By the definition of "yields in one step" in Definition 2.7, we have $|C_i| \leq |C_{i-1}| + 1$ for all $i = 2, \ldots, j$. Unfolding this inequality yields $|C_i| \leq |C_1| + (i-1)$ for all $i = 1, \ldots, j$. Since $|C_1| + (i-1) \leq |C_1| + (j-1) \leq 6 + |x| + p_1(|x|)$ for all $i = 1, \ldots, j$, therefore, $|C_i| \leq 6 + |x| + p_1(|x|)$ for all $i = 1, \ldots, j$. Hence, $\sum_{i=1}^{p_1(|x|)+1} (|C_i| + 1) \leq (p_1(|x|) + 1) \times (6 + |x| + p_1(|x|) + 1)$. As $|\pi| \leq \sum_{i=1}^{j} (|C_i| + 1) \leq \sum_{i=1}^{p_1(|x|)+1} (|C_i| + 1)$, we get:

$$|\pi| \leq \underbrace{(p_1(|x|) + 1) \times (7 + |x| + p_1(|x|))}_{p_2(|x|)}$$

This shows that $|\pi|$ is upperbounded by a polynomial in the length of x. Therefore, $\exists\pi(|\pi| \leq p_2(|x|) \wedge V \text{ accepts } (x,\pi))$ holds.

Now, suppose that there exists a string π such that $|\pi| \leq p_2(|x|)$ and V accepts (x,π). Show that $x \in L$. So, let π_1 be a sequence of configurations such that $|\pi_1| \leq p_2(|x|)$ and V accepts (x,π_1). By the definition of V, we can let $\pi_1 = C_1,\ldots,C_j$, where $C_1 = (s,\triangleright,x), C_j = (yes,w,u)$ and $C_i \xrightarrow{N^1} C_{i+1}$, for all $i = 1,\ldots,j-1$. Hence, $(s,\triangleright,x) \xrightarrow{N^+} (yes,w,u)$. By Definition 2.8, $x \in L$.

($\Leftarrow$) Let V be a polynomial time Turing machine and p_2 be a polynomial such that for all $x \in \Gamma^*$, $x \in L \Leftrightarrow \exists\pi(|\pi| \leq p_2(|x|) \wedge V \text{ accepts } (x,\pi))$ holds. Show that there exists a non-deterministic Turing machine that decides L in polynomial time in the length of the input. Define N to be a non-deterministic Turing machine such that for every input $x \in \Gamma^*$, N behaves as follows:

1. guesses a sequence of configurations π such that $|\pi| \leq p_2(|x|)$;
2. runs V on input (x,π). If V accepts, then so does N, otherwise N rejects.

Let x be an arbitrary string in Γ^*. Clearly, N accepts or rejects x in polynomial time in the length of x, since the first step takes at most $p_2(|x|)$, and V accepts or rejects (x,π) in polynomial time in the length of x.

Suppose that $x \in L$. Hence, $\exists\pi(|\pi| \leq p_2(|x|) \wedge V \text{ accepts } (x,\pi))$ holds. So, let π_1 be a sequence of configurations such that $|\pi_1| \leq p_2(|x|)$ and V accepts (x,π_1). Now, we run N on input x. Let N guess π_1. Hence, N accepts x.

Suppose that $x \notin L$. Hence, $\forall\pi(V \text{ rejects } (x,\pi))$ holds. Therefore, if we run N on input x, then for any sequence of configurations π that N may guess, V will reject (x,π). Therefore, N rejects x. □

Definition 2.11 *Let L and L' be languages such that $L \subseteq \Gamma_1^*$ and $L' \subseteq \Gamma_2^*$, where Γ_1 and Γ_2 are finite alphabets. L is* ***polynomial time reducible to*** *L', denoted $L \leq_P L'$, iff there exists a deterministic Turing machine M and a polynomial p such that for every input string $x \in \Gamma_1^*$, M halts within $p(|x|)$ steps and $M(x) \in L'$ iff $x \in L$. M is called a* ***polynomial time reduction*** *from L to L'.*

Observation 2.12 *The relation $\leq_P$ is reflexive and transitive.*

Definition 2.13 *A language L is* ***NP-complete*** *iff $L \in$ NP and for every language $L' \in$ NP, $L' \leq_P L$.*

Lemma 2.14 *If $L \leq_P L'$ and $L' \in$ NP, then $L \in$ NP.*

Proof: Suppose that $L \leq_P L'$ and $L' \in$ NP. Show that $L \in$ NP. Let M be a polynomial time reduction from L to L' and N' be a non-deterministic Turing machine which decides L' in polynomial time. Define a nondeterministic Turing machine N which decides L as follows.

N on input x:

1. Computes $M(x)$.
2. Runs N' on input $M(x)$.

N obviously runs in polynomial time since its two stages run in polynomial time. Hence, $L \in$ NP. $\square$

Lemma 2.15 *If L is* NP*-complete, $L' \in$ NP and $L \leq_P L'$ then L' is* NP*-complete.*

Proof: Suppose that L is NP-complete, $L' \in$ NP and $L \leq_P L'$. Show that L' is NP-complete. Since L' is already in NP, it suffices to show that for any $L'' \in$ NP, $L'' \leq_P L'$. As L is NP-complete, we get that $L'' \leq_P L$, by Definition 2.13. Since $L \leq_P L'$ and $\leq_P$ is transitive, we get that $L'' \leq_P L'$. $\square$

3 Propositional Proof Complexity

In this section, we give a brief overview of Propositional Proof Complexity and provide definitions that are needed for our purpose. In the first part, we define the language of propositional logic. Then, we relate the NP vs coNP question to the P vs NP question. The search for an efficient proof system for TAUT can be reduced to finding the most powerful of all propositional proof systems, in terms of efficiency, which is an optimal proof system. Within that section, we also define the notion of optimal proof system, that is going to be at the heart of this dissertation. In fact, if one proves the existence of an optimal proof system P, then proving NP is different from coNP boils down to showing that P is not efficient. Later, in that section, we introduce Frege and Substitution Frege systems, as they are needed for the proof of the main theorem of this thesis. Then, we present some results and open problems in propositional proof complexity.

3.1 Propositional Logic

Our exposition of propositional logic follows [CN10].

3.1.1 Syntax and Semantics of Propositional Logic

The language of propositional logic consists of: the logical constants $\top$ (for True) and $\bot$ (for False), a countable set $V = \{p_0, p_1, \ldots\}$ of propositional variables, the logical connectives $\neg$, $\vee$, $\wedge$ and parentheses (,).

Definition 3.1 *Define* ***propositional formulae*** *(or* ***formulae*** *for short) inductively as follows:*

(PL1). $\top, \bot$ *and* p_i *are atomic formulae, for any* $i \geq 0$.

(PL2). *If* φ *and* ψ *are formulae, then so are* $\neg\varphi$, $(\varphi \vee \psi)$ *and* $(\varphi \wedge \psi)$.

The set of all well-formed propositional formulae will be denoted by PL. Propositional formulae will be denoted by $\varphi, \psi, \ldots$, possibly subscripted.

Definition 3.2 *A formula φ is said to be* ***closed*** *if it doesn't contain propositional variables.*

Notation $(\varphi \to \psi)$ stands for $(\neg\varphi \vee \psi)$ and $(\varphi \leftrightarrow \psi)$ for $((\varphi \to \psi) \wedge (\psi \to \varphi))$. Also, we write $\bigwedge_{i=1}^{n} \varphi_i$ for $\varphi_1 \wedge \ldots \wedge \varphi_n$ and $\bigvee_{i=1}^{n} \varphi_i$ for $\varphi_1 \vee \ldots \vee \varphi_n$.

Definition 3.3 *Define a **truth assignment** to be a mapping from V to $\{1,0\}$, where* 1 *denotes* True *and* 0 *denotes* False*. Given a truth assignment τ, the truth value of a formula φ, denoted φ^τ, is defined inductively as follows:*

1. $\top^\tau = 1$, $\bot^\tau = 0$ *and* $(p_i)^\tau = \tau(p_i)$;

2. $(\neg\psi)^\tau = 1 - \psi^\tau$; $(\varphi \wedge \psi)^\tau = min\{\varphi^\tau, \psi^\tau\}$; $(\varphi \vee \psi)^\tau = max\{\varphi^\tau, \psi^\tau\}$.

Definition 3.4 *A truth assignment τ **satisfies** a formula φ, denoted $\tau \models \varphi$, if and only if $\varphi^\tau = 1$.*

Definition 3.5 *Let $\varphi_0, \varphi_1, \ldots, \varphi_k$ be formulas. Then φ_0 is a **logical consequence** of $\{\varphi_1, \ldots, \varphi_k\}$, denoted $\{\varphi_1, \ldots, \varphi_k\} \models \varphi_0$, if and only if for every truth assignment τ, if $(\varphi_1 \wedge \ldots \wedge \varphi_k)^\tau = 1$, then $(\varphi_0)^\tau = 1$.*

3.2 SAT, TAUT and NP vs coNP

Definition 3.6 *A formula φ is **satisfiable** if and only if there exists an assignment τ such that $\tau \models \varphi$ (we denote by* SAT *the set of all satisfiable formulae). φ is a **tautology** if and only if for all assignments τ, $\tau \models \varphi$ (we denote by* TAUT *the set of all tautologies).*

Observation 3.7 *Let φ be a formula. Then $\{\} \models \varphi$ (or simply written as $\models \varphi$) if and only if $\varphi \in$* TAUT.

Notation If L is a language, then denote by $\overline{L}$ the complement of L.

Observation 3.8 *A formula $\varphi \in$* TAUT *if and only if $\neg\varphi \in \overline{\text{SAT}}$.*

Theorem 3.9 [Coo71] SAT *is* NP*-complete.*

Cook is the first to show the existence of an NP-complete language: SAT. Thus, for P to be equal to NP, it has to be that SAT is in P.

Corollary 3.10 $\overline{\text{TAUT}}$ *is* NP*-complete.*

Proof: One way to prove that $\overline{\text{TAUT}}$ is NP-complete is to show that it is in NP and SAT $\leq_P \overline{\text{TAUT}}$. From there, one obtains that $\overline{\text{TAUT}}$ is NP-complete, by Lemma 2.15.

First demonstrate that $\overline{\text{TAUT}} \in$ NP. Observe that $\overline{\text{TAUT}} = \{\phi \mid \neg\phi \in \text{SAT}\}$. Since $\neg\phi$ can easily be computed in polynomial time by a deterministic Turing machine from ϕ, we get that $\overline{\text{TAUT}} \leq_P$ SAT, by Definition 2.11. Thus, $\overline{\text{TAUT}} \in$ NP, by Lemma 2.14.

The proof of SAT $\leq_P \overline{\text{TAUT}}$ uses exactly the same strategy as the proof of $\overline{\text{TAUT}} \leq_P$ SAT, because SAT $= \{\phi \mid \neg\phi \in \overline{\text{TAUT}}\}$. $\square$

Definition 3.11 *A language $L \in$* coNP *if and only if $\overline{L} \in$* NP

Observation 3.12 TAUT $\in$ coNP.

Proposition 3.13 *If* NP $\neq$ coNP, *then* P $\neq$ NP.

Proof: Prove the contrapositive. Observe that P is closed under complementation. Suppose that P = NP. Hence, coNP = $\{\overline{L} \mid L \in \mathrm{P}\}$ = P. Thus, NP = coNP, by assumption. □

Proposition 3.14 NP = coNP *if and only if* TAUT $\in$ NP.

Proof: First observe that $L \leq_P L'$ if and only if $\overline{L} \leq_P \overline{L'}$.

($\Rightarrow$) Suppose that NP = coNP. Show that TAUT is in NP. By Corollary 3.10, we have that $\overline{\text{TAUT}}$ is NP-complete. Hence, $\overline{\text{TAUT}} \in$ NP, by Definition 2.13. It follows that TAUT $\in$ coNP, by Definition 3.11. Therefore, TAUT $\in$ NP, by assumption.

($\Leftarrow$) Suppose that TAUT $\in$ NP. To show that NP = coNP, it suffices to show that for every language in NP, its complement is also in NP. Let L be an arbitrary language in NP. By corollary 3.10, $\overline{\text{TAUT}}$ is NP-complete. Hence, all languages in NP can be polynomially reduced to $\overline{\text{TAUT}}$, by Definition 2.13. In particular, $L \leq_P \overline{\text{TAUT}}$. Therefore, $\overline{L} \leq_P$ TAUT. Since TAUT $\in$ NP (by assumption), we get that $\overline{L} \in$ NP, by Lemma 2.14. □

3.3 Proof System

Definition 3.15 *Define a* ***propositional proof system*** *to be a polynomial time deterministic Turing machine P such that:*

$$\forall x \in \Gamma^*(x \in \text{TAUT} \Leftrightarrow \exists \pi \in \Gamma^*(P \textit{ accepts } (x, \pi)))$$

Sometimes, we will refer to propositional proof systems as just proof systems.

In [CR79], Cook and Reckow defined a propositional proof system to be a polytime computable onto function $f : \Sigma^* \to \text{TAUT}$, for some finite alphabet Σ. A propositional proof system P, as defined in Definition 3.15, can be transformed into a function f satisfying [CR79]'s definition as follows. If P accepts (x, π), then f maps (x, π) to x, else if P rejects (x, π), then f maps (x, π) to $\top$. In the converse direction, one can construct a polytime deterministic Turing machine P such that P accepts (x, π) if and only if $f(\pi) = x$ as follows. Let P' be a polynomial time deterministic Turing machine that computes f. Now, P, on input (x, π), runs $P'(\pi)$. If $P'(\pi) = x$, then P accepts (x, π), otherwise it rejects. Hence, the two definitions are equivalent. Thus, depending on the context, we may use one or the other later.

Note that the runtime of a propositional proof system depends on the length of π.

Definition 3.16 *We say that a propositional proof system P is* ***polynomially bounded*** *iff there exists a polynomial p such that:*

$$\forall x \in \Gamma^*(x \in \text{TAUT} \Leftrightarrow \exists \pi(P \textit{ accepts } (x, \pi) \wedge |\pi| \leq p(|x|)))$$

If P accepts (x, π), then we say that π is a ***P-proof*** *of x. Additionally, if $|\pi| \leq p(|x|)$, then we say that π is a* ***short*** *P-proof of x.*

Theorem 3.17 *A polynomially bounded propositional proof system exists iff* NP = coNP.

Proof: ($\Rightarrow$) Suppose that there exists a polynomially bounded propositional proof system. By the definition of "polynomially bounded" in Definition 3.16, there exists a polynomial time deterministic Turing machine P and a polynomial p such that $\forall x \in \Gamma^*(x \in \text{TAUT} \Leftrightarrow \exists \pi(P \text{ accepts } (x, \pi) \wedge |\pi| \leq p(|x|)))$ holds. This implies that TAUT $\in$ NP by Theorem 2.10. Therefore, NP = coNP by Proposition 3.14.

($\Leftarrow$) Suppose that NP = coNP. Hence, TAUT $\in$ NP by Proposition 3.14. By Theorem 2.10, there exists a polynomial time deterministic Turing machine P and a polynomial p such that $\forall x \in \Gamma^*(x \in \text{TAUT} \Leftrightarrow \exists \pi(|\pi| \leq p(|x|) \wedge P \text{ accepts } ((x, \pi))))$ holds. This implies that a polynomially bounded propositional proof system exists, by Definition 3.16. □

Theorem 3.17 initiated a program of research (called Cook's program by some) aiming at attacking the NP vs coNP problem by proving that stronger and stronger proof systems are not polynomially bounded, until it is established for all proof systems.

Definition 3.18 *Let P_1 and P_2 be propositional proof systems. We say that P_1* ***p-simulates*** *P_2, denoted $P_2 \leq_P P_1$, iff there exists a polynomial time deterministic Turing machine M such that:*

(2) $$\forall x, \pi(P_2 \text{ accepts } (x, \pi) \Rightarrow P_1 \text{ accepts } (x, M(\pi)))$$

We say that P_1 is ***p-equivalent*** *to P_2, denoted $P_1 \equiv_P P_2$, iff they p-simulate each other.*

Note that in Definition 3.18, the notion of P_1 p-simulates P_2 requires the existence of a polynomial-time deterministic Turing machine that translates every P_2-proof π of a tautology φ into a P_1-proof of φ. There is also a weaker notion of p-simulation, called simulation, where the only thing required is the existence of a P_1-proof π' of φ such that $|\pi'| \leq p(|\pi|)$, for some polynomial p. Below, we give a formal definition of the notion of simulation.

Definition 3.19 *Let P' be a propositional proof system. We say that P' is* ***p-optimal*** *iff for all propositional proof systems P, $P \leq_P P'$.*

Definition 3.20 *Let P_1 and P_2 be propositional proof systems. We say that P_1* ***simulates*** *P_2, denoted $P_2 \leq P_1$, iff there exists a polynomial p such that:*

$$\forall \phi, \pi(P_2 \text{ accepts } (\phi, \pi) \Rightarrow \exists \pi'(P_1 \text{ accepts } (\phi, \pi') \wedge |\pi'| \leq p(|\pi|)))$$

We say that P_1 is ***equivalent*** *to P_2, denoted $P_1 \equiv P_2$, iff they simulate each other.*

Definition 3.21 *Let P' be a propositional proof system. We say that P' is* ***optimal*** *iff for all propositional proof systems P, $P \leq P'$.*

Note that if a proof system $P_2 \leq_P P_1$, then $P_2 \leq P_1$. However, the other direction doesn't hold. Therefore, the relation $\leq_P$ is a strict subset of $\leq$. It follows that $\equiv_P$ is also a strict subset of $\equiv$ and a p-optimal proof system is already an optimal proof system.

Definition 3.22 *Let S be a set and $\mathcal{R}$ be a binary relation on S. $\mathcal{R}$ is a* ***quasi-order*** *or* ***pre-order*** *if and only if:*

1. $\forall e \in S(e\mathcal{R}e)$ *(reflexive),*
2. $\forall e_1, e_2, e_3 \in S(e_1\mathcal{R}e_2 \wedge e_2\mathcal{R}e_3 \Rightarrow e_1\mathcal{R}e_3)$ *(transitive).*

Notation We denote by $(S, \mathcal{R})$ the set S equipped with the pre-order $\mathcal{R}$.

Definition 3.23 *Let S be a set and $\mathcal{R}$ be a relation on S. Then $\mathcal{R}$ is an* ***equivalence relation*** *on S if and only if $\mathcal{R}$ is a pre-order on S and $\forall e_1, e_2 \in S(e_1\mathcal{R}e_2 \Rightarrow e_2\mathcal{R}e_1)$ (symmetric).*

Proposition 3.24 *Let P_1 be a propositional proof system. Then $P_1 \leq_P P_1$.*

Proof: Construct a polynomial time Turing machine M_1 which on input π will do nothing but output π. Obviously, if π is a P_1-proof, then $M_1(\pi)$ is a P_1-proof as well. By Definition 3.18, $P_1 \leq_P P_1$. □

Proposition 3.25 *Let P_1 be a propositional proof system. Then $P_1 \leq P_1$.*

Proof: Trivial. □

Proposition 3.26 *Let P_1, P_2 and P_3 be propositional proof systems. If $P_1 \leq_P P_2$ and $P_2 \leq_P P_3$, then $P_1 \leq_P P_3$.*

Proof: Suppose that $P_1 \leq_P P_2$ and $P_2 \leq_P P_3$. By Definition 3.18, let M_1 be a polynomial time Turing machine such that for any P_1-proof π_1 there exists a corresponding P_2-proof $M_1(\pi_1)$ and let M_2 be a polynomial time Turing machine such that for any P_2-proof π_2 there exists a corresponding P_3-proof $M_2(\pi_2)$. Construct a polynomial time Turing machine M_3 which on input π behaves as follows: computes $M_1(\pi)$ and then run M_2 on $M_1(\pi)$. Clearly, if the input of M_3 is a P_1-proof, then the output produced is a P_3-proof. By Definition 3.18, $P_1 \leq_P P_3$. □

Proposition 3.27 *Let P_1, P_2 and P_3 be propositional proof systems. If $P_1 \leq P_2$ and $P_2 \leq P_3$, then $P_1 \leq P_3$.*

Proof: Suppose that $P_1 \leq P_2$ and $P_2 \leq P_3$. By Definition 3.20: there exists a polynomial p such that for every P_1-proof π of a tautology ϕ, there exists a P_2-proof π' of ϕ such that $|\pi'| \leq p(|\pi|)$; there exists a polynomial p such that for every P_2-proof π of a tautology ϕ, there exists a P_3-proof π' of ϕ such that $|\pi'| \leq p(|\pi|)$. Now, we want to show that for every P_1-proof π of a tautology ϕ, there exists a P_3-proof π' of ϕ such that $|\pi'| \leq p(|\pi|)$, for some polynomial p. So, let ϕ be any tautology and π_1 be any P_1-proof of ϕ. Thus, we can let π_2 be a P_2-proof of ϕ such that $|\pi_2| \leq p_1(|\pi_1|)$, for some polynomial p_1. Furthermore, we can let π_3 be a P_3-proof of ϕ such that $|\pi_3| \leq p_2(|\pi_2|)$, for some polynomial p_2. Thus, $|\pi_3| \leq p_3(|\pi_1|)$, where $p_3(|\pi_1|) = p_2(p_1(|\pi_1|))$. By Definition 3.20, $P_1 \leq P_3$. □

The proofs of Propositions 3.24, 3.25, 3.26 and 3.27 show that $\leq_P$ and $\leq$ are pre-orders on the set of all propositional proof systems. The relation $\equiv_P$ and $\equiv$ are obviously equivalence relations, since they are both symmetric by definition.

Definition 3.28 *A **greatest element** of $(S, \mathcal{R})$ is an element $g \in S$ such that for all $e \in S$, $e\mathcal{R}g$.*

Observation 3.29 *Let PPS denote the set of all propositional proof systems. $(PPS, \leq)$ has a greatest element iff there exists an optimal proof system within PPS.*

Proof: ($\Rightarrow$) Suppose that a greatest element exists within $(PPS, \leq)$. Let P be such element. By Definition 3.28, $\forall P' \in PPS(P' \leq P)$. Hence, P is optimal by Definition 3.21.

($\Leftarrow$) Suppose that there exists an optimal proof system within PPS. Let P be such proof system. Hence, $\forall P' \in PPS(P' \leq_P P)$ holds, by Definition 3.21. By Definition 3.28, P is a greatest element within $(PPS, \leq_P)$. □

Note that the existence of an optimal proof system doesn't imply the existence of a p-optimal proof system. However, the existence of a p-optimal proof system implies the existence of an optimal proof system.

3.4 Frege and Substitution Frege Systems

Our exposition of Frege and substitution Frege systems follows [CR79].

3.4.1 Frege Systems

Definition 3.30 *Define a **substitution** σ to be a mapping from the set of propositional variables to the set of propositional formulae. If $\varphi \in$ PL, then denote by $\varphi\sigma$ the result of replacing every variable in φ by its image under σ.*

Lemma 3.31 *Let φ be a propositional formula and σ be a substitution. If $\varphi \in$ TAUT, then $\varphi\sigma \in$ TAUT.*

Proof: We prove the contrapositive. Suppose that $\varphi\sigma \notin$ TAUT. Thus, there exists a truth assignment τ such that $(\varphi\sigma)^\tau = 0$. Let τ' be a truth assignment defined as follows: for every propositional variable p in φ, $\tau'(p) = (p\sigma)^\tau$. Then, one can show by structural induction that for every subformula ψ of φ, $\psi^{\tau'} = (\psi\sigma)^\tau$, in particular $\varphi^{\tau'} = (\varphi\sigma)^\tau = 0$. Therefore, φ is not a tautology. □

Definition 3.32 *A **Frege rule** is a system of propositional formulae of the form*

$$\frac{\varphi_1, \ldots, \varphi_k}{\varphi_0} \tag{3}$$

*such that $\{\varphi_1, \ldots, \varphi_k\} \models \varphi_0$. If $k = 0$, then the rule is called a **Frege axiom scheme**. We shall also write $(\varphi_1, \ldots, \varphi_k)/\varphi_0$ for (3).*

Remark 3.33 *If $(\varphi_1, \ldots, \varphi_k)/\varphi_0$ is a Frege rule, then $\varphi_1 \wedge \ldots \wedge \varphi_k \Rightarrow \varphi_0$ is a tautology.*

Definition 3.34 *Define an **inference system** $\mathscr{F}$ to be a finite set of Frege rules.*

Definition 3.35 *A formula ϕ_0 is* ***inferred from*** *$\phi_1, \ldots, \phi_k$ by the Frege rule $(\varphi_1, \ldots, \varphi_k)/\varphi_0$ if there exists a substitution σ such that for every i from 0 to k, $\phi_i = \varphi_i\sigma$.*

Definition 3.36 *Let $\mathscr{F}$ be an inference system. A* ***Frege proof****, or $\mathscr{F}$-proof for short, of a propositional formula ϕ from Γ (finite set of propositional formulae) is a sequence $\pi = \phi_1 \ldots, \phi_m$ of propositional formulae such that ϕ_m is ϕ and for every i from 1 to m, ϕ_i is either in Γ or inferred from $\phi_{u_1}, \ldots, \phi_{u_k}$ by a rule in $\mathscr{F}$, where $u_1 < \ldots < u_k < i$.*

Notation If Γ is a finite set of propositional formulae, $\mathscr{F}$ an inference system and ϕ a formula, then $\Gamma \vdash_{\mathscr{F}} \phi$ means that there exists an $\mathscr{F}$-proof of ϕ from Γ.

Theorem 3.37 *Let $\mathscr{F}$ be an inference system. Then, for any finite set Γ of propositional formulae and $\phi \in \mathrm{PL}$, if $\Gamma \vdash_{\mathscr{F}} \phi$, then $\Gamma \models \phi$.*

Proof: Let Γ be an arbitrary set of propositional formulae and $\phi \in \mathrm{PL}$. Suppose that $\Gamma \vdash_{\mathscr{F}} \phi$. Show that $\Gamma \models \phi$. Let τ be any truth assignment. Suppose that $\gamma^\tau = 1$ for every $\gamma \in \Gamma$. Let $\pi = \phi_1, \ldots, \phi_m$ be an $\mathscr{F}$-proof of ϕ from Γ. Show by induction on i that $\phi_i^\tau = 1$. If $\phi_i \in \Gamma$, then $\phi_i^\tau = 1$. Suppose that ϕ_i is inferred from $\phi_{u_1}, \ldots, \phi_{u_k}$, where $u_1 < \ldots < u_k < i$, by a Frege rule $(\varphi_1, \ldots, \varphi_k)/\varphi_0$ in $\mathscr{F}$. By Definition 3.35, there exists a substitution σ such that for every j from 1 to k, $\phi_{u_j} = \varphi_j\sigma$ and $\phi_i = \varphi_0\sigma$. Now, there are two subcases to consider. If $k = 0$, then $\phi_i^\tau = 1$. Assume that $k \neq 0$. By induction hypothesis, $(\phi_{u_1} \wedge \ldots \wedge \phi_{u_k})^\tau = 1$. By Lemma 3.31, we have that $\phi_{u_1} \wedge \ldots \phi_{u_k} \Rightarrow \phi_i$ is a tautology. Thus, $\phi_i^\tau = 1$. □

Definition 3.38 *Let $\{\phi_1, \ldots, \phi_l\}$ be an arbitrary set of propositional formulae. An inference system $\mathscr{F}$ is said to be* ***implicationally complete*** *iff for any $\phi \in \mathrm{PL}$, if $\{\phi_1, \ldots, \phi_l\} \models \phi$, then $\{\phi_1, \ldots, \phi_l\} \vdash_{\mathscr{F}} \phi$.*

Definition 3.39 *Let $\mathscr{F}$ be an inference system. We say that $\mathscr{F}$ is a* ***Frege system*** *if and only if $\mathscr{F}$ is implicationally complete.*

Theorem 3.40 *If φ is a closed formula, then either φ or $\neg\varphi$ has a poly-size Frege-proof.*

Proof: Assume that the Frege system under consideration below includes the following Frege rules: $R_1 = (A, B)/A \wedge B$, $R_2 = (A)/A \vee B$, $R_3 = (B)/A \vee B$, $R_4 = (\neg A)/\neg(A \wedge B)$.

Suppose that φ is a closed formula. We show by structural induction on φ that either φ or $\neg\varphi$ has a poly-size Frege-proof. If φ is $\top$ or $\bot$, then it is trivial. Suppose that φ is of the form $\varphi_1 \wedge \varphi_2$. If φ evaluates to True, then φ_1 and φ_2 evaluate to True. By induction hypothesis, they have poly-size Frege-proofs. By R_1, we obtain a Frege-proof of $\varphi_1 \wedge \varphi_2$. Thus, $\varphi_1 \wedge \varphi_2$ has a poly-size Frege-proof. If φ evaluates to False, then either φ_1 or φ_2 evaluates to False. Assume w.l.o.g. that φ_1 evaluates to False. By induction hypothesis, $\neg\varphi_1$ has a poly-size Frege-proof. By R_4, we obtain a Frege-proof of $\neg(\varphi_1 \wedge \varphi_2)$. Thus, $\neg\varphi$ has a poly-size Frege-proof. Similarly, for φ of the form $\varphi_1 \vee \varphi_2$. Suppose that φ is of the form $\neg\psi$. If φ evaluates to True, then ψ evaluates to False. By induction hypothesis, φ has a poly-size Frege-proof. If φ evaluates to False, then $\neg\varphi$, which is ψ, evaluates to True. By induction hypothesis, ψ has poly-size Frege-proof. Thus, $\neg\varphi$ has poly-size Frege-proof. □

Corollary 3.41 *If φ is a closed tautology, then it has a poly-size Frege-proof.*

Corollary 3.41 is a direct implication of Theorem 3.40.

Remark 3.42 *The size of a Frege-proof of a closed tautology ϕ is quadratic in the size of ϕ, since the number of lines in a Frege-proof of ϕ is linear in the length of ϕ and the number of symbols in a line of a Frege-proof is linear in the length of ϕ.*

3.4.2 Substitution Frege systems

Definition 3.43 *Define a* ***substitution Frege system*** *$s\mathscr{F}$ to be a Frege system $\mathscr{F}$ plus the* ***substitution rule*** *$\varphi/\varphi\sigma$, which states that from propositional formula φ infer $\varphi\sigma$, for any substitution σ.*

Definition 3.44 *Let $\mathscr{F}$ be a Frege system. A* ***substitution Frege proof,*** *or $s\mathscr{F}$-proof for short, of $\phi \in \mathrm{TAUT}$ is a sequence $\pi = \phi_1, \ldots, \phi_m$ of propositional formulae such that $\phi_m = \phi$ and for every i from 1 to m, ϕ_i is either inferred from $\phi_{u_1}, \ldots, \phi_{u_k}$, where $u_1 < \ldots < u_k < i$, by a Frege rule in $\mathscr{F}$ or inferred from ϕ_j, where $j < i$, by the substitution rule.*

We can eliminate an application of the substitution rule by repeating the part of the proof before the inference. In such a transformation, these repetitions can be nested and the proof may grow exponentially.

Observe that premises are not allowed in the definition of substitution Frege proofs. If premises were allowed in the definition of substitution Frege proofs, then there exists a $\phi \in \mathrm{PL}$ and some substitution σ such that $\phi \vdash_{s\mathscr{F}} \phi\sigma$ and $\phi \not\models \phi\sigma$. For example, when $\phi = p_1 \wedge p_2$ and σ maps p_1 to itself and maps p_2 to $p_1 \wedge \neg p_1$.

Theorem 3.45 *Let $s\mathscr{F}$ be a substitution Frege system. For any $\phi \in \mathrm{PL}$, if $\vdash_{s\mathscr{F}} \phi$, then $\models \phi$.*

Proof: The proof is similar to the proof of Theorem 3.37. □

3.4.3 Some Results and Open Problems in Proof Complexity

Cook and Reckow were the first to identify Frege and substitution Frege systems in [CR79]. They also identified another class of proof systems, called Extended Frege systems and showed that all Frege systems are p-equivalent. Krajícek and Pudlak showed, in [KP89b], that Extended Frege systems are p-equivalent with Substitution Frege systems.

With regard to the general program of Propositional Proof Complexity, a lot of work has been devoted to proving strong lower bounds on the sizes of proofs of specific tautologies in various proof systems.

Example 3.46 *For any $n \geq 1$, the* ***Pigeonhole Principle*** *states that if $n+1$ pigeons sit in n holes, then there exists a hole with at least two pigeons.*

Definition 3.47 *We define the family of tautologies that formalises the Pigeonhole Principle to be the set:*

$$\text{PHP} = \{\text{PHP}_n^{n+1} : n \geq 1\} \tag{4}$$

where PHP_n^{n+1} *is defined to be:*

$$\left(\bigwedge_{i \leq n} \bigvee_{j<n} p_{\langle i,j \rangle}\right) \to \left(\bigvee_{j<n} \bigvee_{i_1<i_2 \leq n} p_{\langle i_1,j \rangle} \wedge p_{\langle i_2,j \rangle}\right) \tag{5}$$

where $\langle x, y\rangle$ *is defined in Definition* 4.21 *and the intended meaning of* $p_{\langle i,j\rangle}$ *is that pigeon* i *sits in hole* j.

It has first been shown by Haken, in [Hak85], that $\neg\text{PHP}_n^{n+1}$ requires exponential size Resolution refutations. Later, Beame and Pitassi provided an improved lower bound on the sizes of Resolution refutations for $\neg\text{PHP}_n^{n+1}$ in [BP96]. On the other hand, in [CR79, Bus87b], it was shown that a proof of PHP_n^{n+1} in extended Frege has length $O(n^5)$ and has length $O(n^c)$ in Frege for some constant c (fairly small, e.g. $c = 20$) , respectively. It follows that Frege and extended Frege simulate Resolution and not the other way around.

It was originally conjectured that for any $m > n$, PHP_n^m would require exponential size, in n, Resolution refutations. However, this conjecture was shown to be wrong for large values of m [BP98], in particular for $m \geq 2^{\sqrt{nlogn}}$. When $n^2 \leq m < 2^{\sqrt{nlogn}}$, no lower bound was known at all and remained an open problem until [Raz04], where the author has proven that for any m such that $n^2 \leq m < 2^{\sqrt{nlogn}}$, any Resolution refutation of $\neg\text{PHP}_n^m$ is of length $\Omega(2^{n^\epsilon})$, where $\epsilon > 0$ is some global constant.

Definition 3.48 *Let* P *be a propositional proof system. Then a countable family of tautologies* $\{\varphi_i : i \in I\}$ *has polysize* P*-proofs if and only if there exists a polynomial* p *such that for every* $i \in I$, *there exists a* π *such that* $P(\varphi_i, \pi)$ *holds and* $|\pi| \leq p(|\varphi_i|)$.

Definition 3.49 *Let* P *be a propositional proof system. Then a countable family of tautologies* $\{\varphi_i : i \in I\}$ *is* ***hard*** *for* P *if* $\{\varphi_i : i \in I\}$ *doesn't have polysize* P*-proofs.*

At present time, no strong lower bounds are known for Frege, Extended and Substitution Frege systems. What makes it so difficult when trying to prove strong lower bounds for these systems is that there is a lack of hard candidate tautologies [BP01]. Thus, a natural open problem is to find hard tautologies for Frege, Extended and Substitution Frege systems. Some candidate hard tautologies have been suggested in [BBP95] for Frege systems.

These open problems have been collected from [BP01]. In there, one may find many more open problems related to Propositional Proof Complexity in general.

4 Uniform Systems

In this section, we define the notion of the uniform reduct of propositional proof systems (also called uniform systems) [Bec05] using the language of second-order bounded arithmetic. The use of second-order bounded arithmetic is justified by the fact that in our main theorem, we assume that the uniform reduct of a proof system is defined using that language. Our exposition of second-order bounded arithmetic and how we translate second-order bounded arithmetic formulae (Σ_0^B-formulae) into propositional formulae follows [CN10].

4.1 Second-Order Bounded Arithmetic

In second-order bounded arithmetic, there are two kinds of variables: the variables $x, y, z, \ldots$ (possibly subscripted), called **number variables**, that are intended to range over $\mathbb{N}$; the variables $U, V, W, X, Y, Z, \ldots$ (possibly subscripted), called **set** (or **string**) **variables**, that are intended to range over the set of finite subsets of $\mathbb{N}$. We need the first sort (numbers) to measure the length of the second sort (strings). We identify strings with finite subsets of $\mathbb{N}$ (made precise later). Predicate symbols $P, Q, R, \ldots$ can take take arguments of both sorts, and so can function symbols. There are two kinds of functions: the **number functions** and the **string functions**. We use $f, g, h, \ldots$ as meta-symbols for number function symbols; we use $F, G, H, \ldots$ for string function symbols and $a, b, c, \ldots$ for number variables (possibly subscripted).

Definition 4.1 *We define an (n, m)-**ary function symbol** to be a function symbol that takes n arguments of the first sort and m arguments of the second sort. A $(0,0)$-ary number (resp. string) function symbol is called a **number constant symbol** (resp. **string constant symbol**).*

Definition 4.2 *Define $\mathcal{L}_A^2$ to be $\{0, 1, +, \times, ||, =_1, =_2, \leq, \in\}$, where 0 and 1 are number constant symbols; $+$ and $\times$ are $(2,0)$-ary number function symbols; $||$ is a $(0,1)$-ary number function symbol; $=_1$ and $\leq$ are $(2,0)$-ary predicate symbols; $=_2$ is a $(0,2)$-ary predicate symbol; $\in$ is a $(1,1)$-ary predicate symbol.*

Notation We write $=$ for both $=_1$ and $=_2$. It will be clear from the context which is intended. Finally, we will use infix notation when using $+, \times, =, \leq$ and $\in$.

Definition 4.3 *Define $\mathcal{L}_A^2$-**terms** inductively as follows:*

1. *Every number variable is an $\mathcal{L}_A^2$-**number term**.*
2. *Every string variable is an $\mathcal{L}_A^2$-**string term**.*
3. *The symbols $0, 1$ are $\mathcal{L}_A^2$-number terms.*
4. *If t_0 and t_1 are $\mathcal{L}_A^2$-number terms, then so are $(t_0 + t_1)$ and $(t_0 \times t_1)$.*
5. *If T is an $\mathcal{L}_A^2$-string term, then $|T|$ is an $\mathcal{L}_A^2$-number term.*

*An $\mathcal{L}_A^2$-number term is said to be **closed** if it is not built up from rule 1 and 2.*

We will refer to $\mathcal{L}_A^2$-number terms (resp. $\mathcal{L}_A^2$-string terms) as just "number terms" (resp. "string terms"). We often denote number terms by $r, s, t, \ldots$ (possibly subscripted). Note that the only string terms are the string variables and if a string variable X occurs in a number term, then it must occur in a number term of the form $|X|$.

Notation If t is a number term not involving x, then $(\exists x \leq t)\varphi$ stands for $\exists x(x \leq t \wedge \varphi)$ and $(\forall x \leq t)\varphi$ stands for $\forall x(\neg(x \leq t) \vee \varphi)$.

Definition 4.4 *We define Σ_0^B -formulae inductively as follows:*

1. *The logical constants $\top$ (True) and $\perp$ (False) are atomic Σ_0^B-formulae.*
2. *$(t_0 = t_1)$, $(t_0 \leq t_1)$, $t_0 \in X$ and $(X = Y)$ are atomic Σ_0^B-formulae, for number terms t_0, t_1.*
3. *If φ and ψ are Σ_0^B-formulae, then so are $\neg\varphi, \varphi \wedge \psi$ and $\varphi \vee \psi$.*
4. *If φ is a Σ_0^B-formula and x a number variable not occuring in the number term t, then $(\exists x \leq t)\varphi$ and $(\forall x \leq t)\varphi$ are Σ_0^B-formulae.*

Σ_0^B-formulae will often be denoted by $\varphi, \psi, \ldots$ (possibly subscripted).

Definition 4.5 *We define the* ***universal closure*** *of a Σ_0^B-formula φ to be the formula $\forall\varphi$ obtained by adding an unbounded universal quantifier for every free number variable and string variable in φ.*

Notation As in the case of propositional formulae, we write $(\varphi \rightarrow \psi)$ for $(\neg\varphi \vee \psi)$ and $\varphi \leftrightarrow \psi$ for $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. We use the abbreviation $X(t)$ for $t \in X$. Additionally, $x \neq y$ stands for $\neg(x = y)$ and $x < y$ for $x \leq y \wedge x \neq y$. Furthermore, $(\exists x < t)\varphi$ and $(\forall x < t)\varphi$ stand for $(\exists x \leq t)(x \neq t \wedge \varphi)$ and $(\forall x \leq t)(x \neq t \rightarrow \varphi)$, respectively. Finally, when we use $\varphi \in \Sigma_0^B$, we mean that φ is a Σ_0^B-formula.

Definition 4.6 *An occurrence of a number variable x in a Σ_0^B-formula φ is* ***bound*** *if and only if that occurrence of x occurs in a subformula of φ of the form $(\exists x \leq t)\psi$ or $(\forall x \leq t)\psi$. Any number variable occurrence in a Σ_0^B-formula that is not bound is said to be* ***free****.*

Definition 4.7 *Let φ be a Σ_0^B-formula. Then define $\mathrm{FV}(\varphi)$ to be the set of all free number variables in φ; define $\mathrm{SV}(\varphi)$ to be the set of all string variables in φ. φ is said to be* ***closed*** *if $\mathrm{FV}(\varphi) = \emptyset$ and $\mathrm{SV}(\varphi) = \emptyset$.*

Notation In what follows, let $\mathcal{P}_{fin}(\mathbb{N})$ denote the set of all finite subsets of $\mathbb{N}$

We identify a set $S \subseteq \mathbb{N}$ with its characteristic function. Hence, we can use function notation and membership notation interchangeably, as the context demands.

Definition 4.8 *Let S be a finite subset of $\mathbb{N}$ and $w : \mathcal{P}_{fin}(\mathbb{N}) \rightarrow \{0,1\}^*$ be the mapping defined as follows*

$$w(S) = S(n-1) \ldots S(1)S(0),$$

where $n - 1$ is the largest element of S. We define the ***binary representation*** *of S to be $w(S)$.*

Note that the binary representation of the empty set is the empty string.

Since w is an injective mapping, we can identify a finite non-empty subset S of $\mathbb{N}$ with its binary string representation.

Definition 4.9 *The $\mathcal{L}_A^2$-**standard model** $\underline{\mathbb{N}}_2$ consists of the following:*

1. *Two non-empty sets $\mathbb{N}$ and $\mathcal{P}_{fin}(\mathbb{N})$ that are called the universes of $\underline{\mathbb{N}}_2$. Number (resp. string) variables range over $\mathbb{N}$ (resp. $\mathcal{P}_{fin}(\mathbb{N})$).*
2. *The number constant symbols 0 and 1 are interpreted by $0, 1 \in \mathbb{N}$, respectively.*
3. *The number function symbols $+$ and $\times$ are interpreted by the addition and multiplication functions on $\mathbb{N}$, respectively.*
4. *The number function symbol $||$ is interpreted by the function $|S|^{\underline{\mathbb{N}}_2}$, which is defined to be the length of the binary representation of the set S (i.e. $1+$ the largest element of S).*
5. *The predicate symbol $=_1$ (resp. $=_2$) is always interpreted as the true equality relation on $\mathbb{N}$ (resp. $\mathcal{P}_{fin}(\mathbb{N})$).*
6. *The predicate symbols $\leq, \in$ get their usual interpretations.*

Definition 4.10 *An **object assignment** consists of a mapping from the number variables to $\mathbb{N}$ and a mapping from the string variables to $\mathcal{P}_{fin}(\mathbb{N})$.*

Notation Let α be an object assignment. Then we write $\alpha(x)$ for the object in $\mathbb{N}$ assigned to x by α and $\alpha(X)$ for the object in $\mathcal{P}_{fin}(\mathbb{N})$ assigned to X by α. If $m \in \mathbb{N}$, then $\alpha(m/x)$ is the same as α except that it maps x to m. If $M \in \mathcal{P}_{fin}(\mathbb{N})$, then $\alpha(M/X)$ is the same as α except that it maps X to M.

Definition 4.11 *Let α be an object assignment. Then define for each number term t (resp. string variable X) its value $t^{\underline{\mathbb{N}}_2}[\alpha] \in \mathbb{N}$ (resp. $X^{\underline{\mathbb{N}}_2}[\alpha] \in \mathcal{P}_{fin}(\mathbb{N})$) in $\underline{\mathbb{N}}_2$ under α inductively as follows:*

1. *$x^{\underline{\mathbb{N}}_2}[\alpha]$ is $\alpha(x)$.*
2. *$X^{\underline{\mathbb{N}}_2}[\alpha]$ is $\alpha(X)$.*
3. *$0^{\underline{\mathbb{N}}_2}[\alpha]$ and $1^{\underline{\mathbb{N}}_2}[\alpha]$ are the natural numbers 0 and 1, respectively.*
4. *If t_0, t_1 are number terms, then $(t_0 + t_1)^{\underline{\mathbb{N}}_2}[\alpha]$ is $(t_0^{\underline{\mathbb{N}}_2}[\alpha] + t_1^{\underline{\mathbb{N}}_2}[\alpha])$.*
5. *If t_0, t_1 are number terms, then $(t_0 \times t_1)^{\underline{\mathbb{N}}_2}[\alpha]$ is $(t_0^{\underline{\mathbb{N}}_2}[\alpha] \times t_1^{\underline{\mathbb{N}}_2}[\alpha])$.*
6. *$|X|^{\underline{\mathbb{N}}_2}[\alpha]$ is $|\alpha(X)|^{\underline{\mathbb{N}}_2}$, i.e. the length of the binary representation of $\alpha(X)$.*

Notation Note that for a closed number term t, we can just write $t^{\underline{\mathbb{N}}_2}$.

Definition 4.12 *Let α be an object assignment. Then define, for each Σ_0^B-formula φ, the relation $\underline{\mathbb{N}}_2 \models \varphi[\alpha]$ ($\underline{\mathbb{N}}_2$ satisfies φ under α) by structural induction on φ:*

1. *$\underline{\mathbb{N}}_2 \models \top$ and $\underline{\mathbb{N}}_2 \not\models \bot$.*
2. *If t_0, t_1 are number terms, then $\underline{\mathbb{N}}_2 \models (t_0 = t_1)[\alpha]$ iff $t_0^{\underline{\mathbb{N}}_2}[\alpha] = t_1^{\underline{\mathbb{N}}_2}[\alpha]$.*
3. *If t_0, t_1 are number terms, then $\underline{\mathbb{N}}_2 \models (t_0 \leq t_1)[\alpha]$ iff $t_0^{\underline{\mathbb{N}}_2}[\alpha] \leq t_1^{\underline{\mathbb{N}}_2}[\alpha]$.*

4. $\underline{\mathbb{N}}_2 \models (X = Y)[\alpha]$ *iff* $\alpha(X) = \alpha(Y)$.

5. *If t is a number term, then* $\underline{\mathbb{N}}_2 \models X(t)[\alpha]$ *iff* $t^{\underline{\mathbb{N}}_2}[\alpha] \in \alpha(X)$.

6. $\underline{\mathbb{N}}_2 \models (\neg\psi)[\alpha]$ *iff* $\underline{\mathbb{N}}_2 \not\models \psi[\alpha]$.

7. $\underline{\mathbb{N}}_2 \models (\varphi_0 \star \varphi_1)[\alpha]$ *iff* $\underline{\mathbb{N}}_2 \models \varphi_0[\alpha] \star \underline{\mathbb{N}}_2 \models \varphi_1[\alpha]$, *for* $\star \in \{\wedge, \vee\}$.

8. *If t is a number term not involving x, then* $\underline{\mathbb{N}}_2 \models ((\exists x \leq t)\psi)[\alpha]$ *iff* $\underline{\mathbb{N}}_2 \models \psi[\alpha(m/x)]$, *for some* $m \leq t^{\underline{\mathbb{N}}_2}[\alpha]$.

9. *If t is a number term not involving x, then* $\underline{\mathbb{N}}_2 \models ((\forall x \leq t)\psi)[\alpha]$ *iff* $\underline{\mathbb{N}}_2 \models \psi[\alpha(m/x)]$ *for all* $m \leq t^{\underline{\mathbb{N}}_2}[\alpha]$.

A Σ_0^B-formula φ is said to be ***valid*** *if and only if $\underline{\mathbb{N}}_2 \models \varphi[\alpha]$, for every object assignment α.*

Notation If φ is closed, then we can just write $\underline{\mathbb{N}}_2 \models \varphi$ instead of $\underline{\mathbb{N}}_2 \models \varphi[\alpha]$.

Notation If $\vec{X}$ is a vector of string variables $X_0, \dots, X_{n-1}$, then $|\vec{X}|$ denotes the vector $|X_0|, \dots, |X_{n-1}|$. Similarly for $|\vec{S}|$, where $\vec{S}$ is a vector of sets.

Notation When writing $\varphi(\vec{x}, \vec{X})$, we mean that $\mathrm{FV}(\varphi) \cup \mathrm{SV}(\varphi) \subseteq \{\vec{x}, \vec{X}\}$. Also, when writing $t(\vec{x}, |\vec{X}|)$, we mean that the set of all variables (number and string variables) in t is a subset or equal to $\{\vec{x}, \vec{X}\}$ and string variables are only occuring in the form $|X_i|$.

Definition 4.13 *Let $\vec{x} = x_0, \dots, x_{k-1}$, $\vec{s}$ be a vector of number terms $s_0, \dots, s_{k-1}$, $t(\vec{x})$ be a number term and $\varphi(\vec{x})$ be a Σ_0^B-formula. Then denote by $t(\vec{s})$ the result of substituting every occurrence of x_i in t by s_i and denote by $\varphi(\vec{s})$ the result of replacing every free occurrence of x_i in φ by s_i.*

Definition 4.14 *Let $\vec{X} = X_0 \dots X_{k-1}$, $\vec{s}$ be a vector of number terms $s_0, \dots, s_{k-1}$ and $t(|\vec{X}|)$ be a number term. Then denote by $t(\vec{s})$ the result of substituting every occurrence of $|X_i|$ in t by s_i.*

Definition 4.15 *Let $\varphi(x)$ be a Σ_0^B-formula and t be a number term. Then t is said to be* ***freely substitutable*** *for x in φ if and only if for any variable y in t, for every occurence of x in φ, x is not in a subformula of φ of the form $(\forall y \leq t_0)\psi$ or $(\exists y \leq t_0)\psi$.*

Let $\vec{x} = x_0, \dots, x_{k-1}$, $\vec{t}$ be a vector of number terms $t_0, \dots, t_{k-1}$ and $\varphi(\vec{x})$ be a Σ_0^B-formula. From now on, we shall write $\varphi(\vec{t})$ if and only if t_i is freely substitutable for x_i in φ.

4.2 Translating Σ_0^B-formulae

In this section, we are going to show how to translate each Σ_0^B-formula $\varphi(\vec{x}, \vec{X})$ into a family

$$||\varphi(\vec{x}, \vec{X})|| = \{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\}$$

of propositional formulae, where n_i is intended to be the length of X_i. For that, we introduce the propositional variables $p_0^{X_0}, p_1^{X_0}, \dots, p_0^{X_1}, p_1^{X_1}, \dots$ where the intended meaning of $p_j^{X_i}$ is $X_i(j)$.

Definition 4.16 *For every $n \in \mathbb{N}$, define $\underline{n}$, called the* **numeral** *for n, inductively as follows:*

$$\underline{0} = 0, \underline{1} = 1,$$
$$\underline{n+1} = (\underline{n} + 1) \text{ for } n \geq 1.$$

For example, the numeral for 4 is $(((1+1)+1)+1)$. The numerals $0, 1, (1+1), ((1+1)+1), \ldots$ for $0, 1, 2, 3, \ldots$, respectively, will be denoted by $0, 1, 2, 3, \ldots$.

Definition 4.17 *Let $\varphi(\vec{x}, \vec{X})$ be a Σ_0^B-formula, $\vec{m}$ and $\vec{n}$ be in $\mathbb{N}$. Define $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}]$ inductively as follows:*

1. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $\top$ or $\bot$, then $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \varphi(\underline{\vec{m}}, \vec{X})$*
2. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $t(\underline{\vec{m}}, |\vec{X}|) = u(\underline{\vec{m}}, |\vec{X}|)$, then*

$$\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] \quad =_{df} \quad \begin{cases} \top & \text{if } t(\underline{\vec{m}}, \underline{\vec{n}})^{\underline{\mathbb{N}}_2} = u(\underline{\vec{m}}, \underline{\vec{n}})^{\underline{\mathbb{N}}_2} \\ \bot & \text{otherwise} \end{cases}$$

3. *Similarly if $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $t(\underline{\vec{m}}, |\vec{X}|) \leq u(\underline{\vec{m}}, |\vec{X}|)$.*
4. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $X_i = X_j$, then there are two cases to consider. If $i = j$, then $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \top$. Else if $i \neq j$, then we reduce the task of translating $\varphi(\underline{\vec{m}}, \vec{X})$ to translating its defining axiom $|X_i| = |X_j| \wedge \forall x < |X_i|(X_i(x) \leftrightarrow X_j(x))$. Note that the defining axiom of $X_i = X_j$ is a Σ_0^B-formula such that it doesn't contain any free number variable and it doesn't contain any subformula of the same form as $\varphi(\underline{\vec{m}}, \vec{X})$.*
5. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $X_i(t(\underline{\vec{m}}, |\vec{X}|))$, then set $j = t(\underline{\vec{m}}, \underline{\vec{n}})^{\underline{\mathbb{N}}_2}$ and*

$$\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] \quad =_{df} \quad \begin{cases} p_j^{X_i} & \text{if } j < n_i - 1 \\ \top & \text{if } j = n_i - 1 \\ \bot & \text{otherwise} \end{cases}$$

Observe that for $n_i = 0$, we have that $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \bot$.

6. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $\neg\psi(\underline{\vec{m}}, \vec{X})$, then $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \neg\psi(\underline{\vec{m}}, \vec{X})[\vec{n}]$.*
7. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $\varphi_0(\underline{\vec{m}}, \vec{X}) \star \varphi_1(\underline{\vec{m}}, \vec{X})$, then*

$$\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \varphi_0(\underline{\vec{m}}, \vec{X})[\vec{n}] \star \varphi_1(\underline{\vec{m}}, \vec{X})[\vec{n}],$$

for $\star \in \{\wedge, \vee\}$.

8. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $(\exists y \leq t(\underline{\vec{m}}, |\vec{X}|))\psi(y, \underline{\vec{m}}, \vec{X})$, then*

$$\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \bigvee_{i=0}^{j} \psi(\underline{i}, \underline{\vec{m}}, \vec{X})[\vec{n}]$$

where $j = t(\underline{\vec{m}}, \underline{\vec{n}})^{\underline{\mathbb{N}}_2}$.

9. *If $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $(\forall y \leq t(\underline{\vec{m}}, |\vec{X}|))\psi(y, \underline{\vec{m}}, \vec{X})$, then*

$$\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] =_{df} \bigwedge_{i=0}^{j} \psi(\underline{i}, \underline{\vec{m}}, \vec{X})[\vec{n}]$$

where $j = t(\underline{\vec{m}}, \underline{\vec{n}})^{\underline{\mathbb{N}}_2}$.

In [CN10], the authors included, as part of the translation, some pruning. Since that pruning isn't relevant for our purpose, we decided to not include it for the sake of simplicity and readability. Also, note that $\varphi(\underline{m}, \vec{X})[\vec{n}]$, where $\vec{X} = X_0, \ldots, X_{l-1}$, has variables $p_0^{X_0}, p_1^{X_0}, \ldots, p_{n_0-2}^{X_0}, \ldots, p_0^{X_{l-1}}, \ldots$ and $p_{n_{l-2}}^{X_{l-1}}$.

Notation If $\vec{m}$ is a vector of natural numbers $m_0, \ldots, m_{k-1}$ and $\vec{S}$ is a vector of sets $S_0, \ldots, S_{l-1}$, then $\vec{m} \in \mathbb{N}$ denotes $m_0, \ldots, m_{k-1} \in \mathbb{N}$, $|\vec{S}| = \vec{n}$ denotes $|S_0| = n_0, \ldots, |S_{l-1}| = n_{l-1}$ and $\vec{S} \subseteq \mathbb{N}$ denotes $S_0, \ldots, S_{l-1} \subseteq \mathbb{N}$. Furthermore, we have that $\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})$ stands for

$$\alpha(m_0/x_0, \ldots, m_{k-1}/x_{k-1}, S_0/X_0, \ldots, S_{l-1}/X_{l-1}).$$

Lemma 4.18 *Let $\vec{m} \in \mathbb{N}$, $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$ and $\varphi(\vec{x}, \vec{X})$ be a Σ_0^B-formula. Then we have that $\underline{\mathbb{N}}_2 \models \varphi(\vec{x}, \vec{X})[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})]$ if and only if $\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})]$.*

Proof: The proof is by structural induction on $\varphi(\vec{x}, \vec{X})$. We only cover a few interesting cases. First, observe that

$$t(\vec{x}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})] = t(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]. \tag{6}$$

Additionally, observe that

(7) $t(\vec{x}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})] \in S_i$ if and only if $t(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})] \in S_i$.

1. Consider the case when $\varphi(\vec{x}, \vec{X})$ is of the form $t(\vec{x}, |\vec{X}|) = u(\vec{x}, |\vec{X}|)$. By Definition 4.12, we have that $\underline{\mathbb{N}}_2 \models \varphi(\vec{x}, \vec{X})[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})]$ is equivalent to

$$t(\vec{x}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})] = u(\vec{x}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})],$$

which is equivalent to

$$t(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})] = u(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})],$$

by (6). That is equivalent to

$$\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})],$$

by Definition 4.12.

2. Consider the case when $\varphi(\vec{x}, \vec{X})$ is of the form $X_i(t(|\vec{x}, |\vec{X}|))$. Then we have that

$$\begin{array}{ll} \underline{\mathbb{N}}_2 \models \varphi(\vec{x}, \vec{X})[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})] & \\ \Leftrightarrow t(\vec{x}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x}, \vec{S}/\vec{X})] \in S_i & \text{(by Definition 4.12)} \\ \Leftrightarrow t(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})] \in S_i & \text{(by (7))} \\ \Leftrightarrow \underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})] & \text{(by Definition 4.12)} \end{array}$$

3. Consider the case when $\varphi(\vec{x},\vec{X})$ is of the form $(\exists y \leq t(\vec{x},|\vec{X}|)\psi(y,\vec{x},\vec{X})$. By Definition 4.12, we have that

$$\underline{\mathbb{N}}_2 \models \varphi(\vec{x},\vec{X})[\alpha(\vec{m}/\vec{x},\vec{S}/\vec{X})]$$

is equivalent to

$$\underline{\mathbb{N}}_2 \models \psi(y,\vec{x},\vec{X})[\alpha(i/y,\vec{m}/\vec{x},\vec{S}/\vec{X})],$$

for some $i \leq t(\vec{x},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{m}/\vec{x},\vec{S}/\vec{X})]$. That is equivalent to

$$\underline{\mathbb{N}}_2 \models \psi(\underline{i},\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})],$$

for some $i \leq t(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})$, by induction hypothesis. By Definition 4.12, there exists an $i \leq t(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]$ such that

$$\underline{\mathbb{N}}_2 \models \psi(\underline{i},\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})]$$

if and only if

$$\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})].$$

□

In what follows, let $\tau_{\vec{S}}$, where $\vec{S} = S_0,\dots,S_l \subseteq \mathbb{N}$ and $|S_i| = n_i$, be any truth assignment such that for every i from 0 to l and k from 0 to $n_i - 2$, if $S_i(k)$ holds, then $\tau_{\vec{S}}(p_k^{X_i}) = 1$, and if $S_i(k)$ doesn't hold, then $\tau_{\vec{S}}(p_k^{X_i}) = 0$.

Lemma 4.19 *Let $\vec{m} \in \mathbb{N}$, $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$, $\varphi(\vec{x},\vec{X})$ be a Σ_0^B-formula and $\alpha(\vec{S}/\vec{X})$ be an object assignment. Then we have that*

$$\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})] \text{ if and only if } (\varphi(\underline{\vec{m}},\vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1.$$

Proof: The proof is by structural induction on $\varphi(\underline{\vec{m}},\vec{X})$. Again, we only cover a few interesting cases. First, observe that

(8) $$t(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})] = t(\underline{\vec{m}},\underline{\vec{n}})^{\underline{\mathbb{N}}_2}.$$

- Consider the case when $\varphi(\underline{\vec{m}},\vec{X})$ is of the form $t(\underline{\vec{m}},|\vec{X}|) = u(\underline{\vec{m}},|\vec{X}|)$. Set $i = t(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]$ and $j = u(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]$. Then we have that

  $\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})]$
  $\Leftrightarrow i = j$ (by Definition 4.12)
  $\Leftrightarrow t(\underline{\vec{m}},\underline{\vec{n}})^{\underline{\mathbb{N}}_2} = u(\underline{\vec{m}},\underline{\vec{n}})^{\underline{\mathbb{N}}_2}$ (by (8))
  $\Leftrightarrow \varphi(\underline{\vec{m}},\vec{X})[\vec{n}] =_{df} \top$ (by Definition 4.17)
  $\Leftrightarrow (\varphi(\underline{\vec{m}},\vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1$ (by Definition 3.3)

- Consider the case when $\varphi(\underline{\vec{m}},\vec{X})$ is of the form $X_i(t(\underline{\vec{m}},|\vec{X}|))$. Set $j = t(\underline{\vec{m}},|\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]$. Then we have that

  $\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}},\vec{X})[\alpha(\vec{S}/\vec{X})]$
  $\Leftrightarrow S_i(j)$ (by Definition 4.12)
  $\Leftrightarrow \tau_{\vec{S}}(p_j^{X_i}) = 1$
  $\Leftrightarrow (\varphi(\underline{\vec{m}},\vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1$ (by Definition 4.17)

- Consider the case when $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $\neg\psi(\underline{\vec{m}}, \vec{X})$. Then we have that

$$\begin{array}{ll} \underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})] & \\ \Leftrightarrow \underline{\mathbb{N}}_2 \not\models \psi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})] & \text{(by Definition 4.12)} \\ \Leftrightarrow (\psi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 0 & \text{(by IH)} \\ \Leftrightarrow (\neg\psi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1 & \text{(by Definition 3.3)} \\ \Leftrightarrow ((\neg\psi(\underline{\vec{m}}, \vec{X}))[\vec{n}])^{\tau_{\vec{S}}} = 1 & \text{(by Definition 4.17)} \\ \Leftrightarrow (\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1 & \end{array}$$

- Consider the case when $\varphi(\underline{\vec{m}}, \vec{X})$ is of the form $\exists y \leq t(\underline{\vec{m}}, |\vec{X}|))\psi(y, \underline{\vec{m}}, \vec{X}$. Set $j = t(\underline{\vec{m}}, |\vec{X}|)^{\underline{\mathbb{N}}_2}[\alpha(\vec{S}/\vec{X})]$. Then we have that

$$\begin{array}{ll} \underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})] & \\ \Leftrightarrow \exists i \leq j(\underline{\mathbb{N}}_2 \models \psi(y, \underline{\vec{m}}, \vec{X})[\alpha(i/y, \vec{S}/\vec{X})]) & \text{(by Definition 4.12)} \\ \Leftrightarrow \exists i \leq j(\underline{\mathbb{N}}_2 \models \psi(\underline{i}, \underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})]) & \text{(by Lemma 4.18)} \\ \Leftrightarrow \exists i \leq j((\psi(\underline{i}, \underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1) & \text{(by IH)} \\ \Leftrightarrow \left(\bigvee_{i=0}^{j} \psi(\underline{i}, \underline{\vec{m}}, \vec{X})[\vec{n}] \right)^{\tau_{\vec{S}}} = 1 & \text{(by Definition 3.3)} \\ \Leftrightarrow (\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1 & \text{(by Definition 4.17)} \end{array}$$

□

Theorem 4.20 *Let α be an object assignment, $\varphi(\vec{x}, \vec{X})$ be a Σ_0^B-formula and $\vec{m}, \vec{n} \in \mathbb{N}$. Then we have that $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}]$ is a tautology if and only if the following holds:*

$$\underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})] \tag{9}$$

for any $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$.

Proof: ($\Rightarrow$) Suppose that $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}]$ is a tautology. Show that for any $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$, $\underline{\mathbb{N}}_2$ satisfies $\varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})]$. Let $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$. By assumption, $(\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{S}}} = 1$. By Lemma 4.19, $\underline{\mathbb{N}}_2$ satisfies $\varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})]$.

($\Leftarrow$) Suppose that $\underline{\mathbb{N}}_2$ satisfies $\varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{S}/\vec{X})]$ for any $\vec{S} \subseteq \mathbb{N}$ such that $|\vec{S}| = \vec{n}$. Show that $\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}]$ is a tautology, i.e. for any truth assignment τ, $(\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau} = 1$. For the sake of contradiction, suppose that there exists a truth assignment τ_0 such that $(\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_0} = 0$. Let $\vec{M} \subseteq \mathbb{N}$ such that $|\vec{M}| = \vec{n}$ and $\tau_{\vec{M}} = \tau_0$. Hence, $(\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}])^{\tau_{\vec{M}}} = 0$. By Lemma 4.19, we have that $\underline{\mathbb{N}}_2$ doesn't satisfy $\varphi(\underline{\vec{m}}, \vec{X})[\alpha(\vec{M}/\vec{X})]$, which contradicts our original assumption. □

Definition 4.21 *Define $\langle x_0, x_1 \rangle$, called* ***the pairing function****, as the following $\mathcal{L}_A^2$-term:*

$$(x_0 + x_1) \times (x_0 + x_1 + 1) + 2 \times x_1$$

Observe that the pairing function is a one-to-one function.

Recall that $(\varphi \to \psi)$ and $(\exists x < t)\varphi$ stand for $(\neg\varphi \vee \psi)$ and $(\exists x \leq t)(x \neq t \wedge \varphi)$, respectively. When writing $X(x,y)$, we mean $X(\langle x,y\rangle)$, where $\langle x,y\rangle$ is the pairing function. We now show how to obtain an equivalent form of PHP (Definition 3.47) from a Σ_0^B-formula $\mathrm{PHP}(z,X)$, where z stands for the number of holes and X is intended to be a two-dimensional Boolean array such that $X(x,y)$ holds if and only if pigeon x sits in hole y, for $x \leq z$ and $y < z$. First, define $\mathrm{PHP}(z,X)$ to be the following Σ_0^B-formula:

$$
\begin{aligned}
&(\forall x \leq z)(\exists y < z)X(x,y) \to \\
&(\exists y < z)(\exists x_0, x_1 \leq z)(x_0 < x_1 \wedge X(x_0,y) \wedge X(x_1,y))
\end{aligned}
\tag{10}
$$

By translating $\mathrm{PHP}(z,X)$ into a propositional formula (with the appropriate length for X) and then applying a suitable substitution to the resulting formula, we obtain a formula which is equivalent to PHP_n^{n+1} as follows. For every $n \geq 1$, $\mathrm{PHP}(\underline{n},X)[2 + \langle n, n-1\rangle]\sigma$ can be proven to be equivalent to PHP_n^{n+1} by a short Resolution proof (all that is needed to be done is some pruning), where $\sigma : \{p^X_{\langle i,j\rangle} : i \leq n \text{ and } j \leq n-1\} \to \{p_{\langle i,j\rangle} : i \leq n \text{ and } j \leq n-1\}$ is a substitution and is defined by $\sigma(p^X_{\langle i,j\rangle}) = p_{\langle i,j\rangle}$.

4.3 The Uniform Reduct of a Proof System

Recall that a countable family of tautologies $\{\varphi_i : i \in I\}$ has poly-size P-proofs, where P is a propositional proof system, if and only if there exists a polynomial p such that for every $i \in I$, there exists a π such that P accepts (φ_i, π) and $|\pi| \leq p(|\varphi_i|)$.

Definition 4.22 [Bec05] *Let P be a propositional proof system. Then define the* ***uniform reduct*** *of P to be the set*

$$\mathrm{U}_P = \{\varphi(\vec{x},\vec{X}) \in \Sigma_0^B : \{\varphi(\underline{\vec{m}},\vec{X})[\vec{n}] : \vec{m},\vec{n} \in \mathbb{N}\} \text{ has polysize } P\text{-proofs}\}$$

The uniform reduct of a proof system will also be called a ***uniform system.***

Observation 4.23 *Let P be a propositional proof system. Then P is not polynomially bounded if there exists a valid Σ_0^B-formula $\varphi(\vec{x},\vec{X})$ such that $\varphi(\vec{x},\vec{X}) \notin \mathrm{U}_P$.*

Proof: Suppose that there exists a valid Σ_0^B-formula $\varphi(\vec{x},\vec{X})$ such that $\varphi(\underline{\vec{m}},\vec{X}) \notin \mathrm{U}_P$. Hence, $\{\varphi(\underline{\vec{m}},\vec{X})[\vec{n}] : \vec{m},\vec{n} \in \mathbb{N}\}$ doesn't have polysize P-proofs by Definition 4.22. Therefore, for every polynomial p, there exists a $\psi \in \{\varphi(\underline{\vec{m}},\vec{X})[\vec{n}] : \vec{m},\vec{n} \in \mathbb{N}\}$ such that for every π, either π is not a P-proof of ψ or $|\pi| \not\leq p(|\psi|)$, by Definition 3.48. Thus, P is not polynomially bounded, by Definition 3.16. □

A reason for studying uniform systems is that we might be able to prove lower bounds by identifying properties which distinguish uniform systems from the set of all true Σ_0^B-formulae. In [Bec05], Beckmann studied the arithmetic complexity of uniform systems and observed that a given uniform reduct is not in some certain arithmetic complexity class would imply a super-polynomial lower bound of the underlying propositional proof system. He also investigated whether uniform systems are closed under the typical inference rules of Hilbert style proof systems, i.e. under modus ponens and generalisation.

A natural open problem for uniform systems is to look for properties of uniform systems which might help distinguish uniform systems from the set of all true Σ_0^B-formule, denoted $\mathrm{TRUE}_{\Sigma_0^B}$ (this is the last open problem listed in [Bec05]).

Frege systems are p-equivalent with the propositional part PK of Gentzen's sequent-based proof system LK. Another natural open problem for uniform systems would be to prove if U_{PK} is equal or not to $\mathrm{TRUE}_{\Sigma_0^B}$.

5 Uniform Systems vs Optimal Proof Systems

We finally come to the main body of this thesis. In this last section, we carry out a detailed proof of the equivalence between the existence of an optimal proof system and the existence of a propositional proof system whose uniform reduct equals the set of all true Σ_0^B-formulae. As a preliminary to that, we first show how to Σ_0^B-formulate the reflection principle for a propositional proof system.

5.1 The Reflection Principle for a Proof System

In order to Σ_0^B-formulate the reflection principle for a propositional proof system, we first have to show how to encode propositional formulae, but this time in Polish notation. Then, we define how to encode truth assignments to those formulae. After that, we show how to encode polytime Turing machine computations.

5.1.1 Encoding Polish Propositional Formulae

Polish propositional formulae are propositional formulae, but in prefix notation and where propositional variables will have their indices written in unary. For example, p_i is written as $p11\ldots1$ with i many 1's.

Definition 5.1 ***Polish propositional formulae*** *(or* ***Polish formulae*** *for short) are over the alphabet*

$$\Sigma = \{p, 1, \neg, \wedge, \vee\}$$

and defined inductively as follows:

1. *Every propositional variable is an atomic Polish formula.*
2. *If φ is a Polish formula, then so is $\neg\varphi$.*
3. *If φ, ψ are Polish formulae, then so are $\star\varphi\psi$, where $\star \in \{\wedge, \vee\}$.*

Definition 5.2 *Define the* ***subformulae*** *of a Polish formula φ inductively as follows:*

1. *If φ is of the form $p11\ldots1$, then its only subformula is $p11\ldots1$.*
2. *If φ is of the form $\neg\psi$, then its subformulae are the subformulae of ψ plus φ itself.*

3. *If φ is of the form $\star\varphi_0\varphi_1$, where $\star \in \{\wedge, \vee\}$, then the subformulae of φ is the subformulae of φ_0 plus the subformulae of φ_1 plus φ itself.*

Note that the subformulae of a Polish formula are Polish formulae. For example, the subformulae of the Polish formula $\wedge p11\neg\vee p11p1$ are the following Polish formulae: $p1, p11, \vee p11p1, \neg\vee p11p1, p111$ and $\wedge p111\neg\vee p11p1$ itself. Also, note that if $s = s_n \dots s_1 s_0$ is a Polish formula, where $s_i \in \Sigma$ for every i from 0 to n, and $s_j \dots s_k$ is a subformula of s, where $n \geq j \geq k \geq 0$, then the following statement holds: if $k > 0$, then s_{k-1} is different from the symbol 1 (otherwise s is not a well-formed Polish formula).

Notation Let $s = s_n \dots s_1 s_0$ be a string over some alphabet and $n \geq j \geq k \geq 0$. Then denote by $s[j,k]$ the substring $s_j \dots s_{k+1} s_k$ of s. For $n \geq i \geq 0$, let $s[i]$ denote s_i. Thus, if $j = k$, then $s[j,k]$ is the same as $s[j]$.

Observation 5.3 *If $s = s_n \dots s_0$ is a string over Σ such that s is a Polish formula, then for every $i \leq n$, if s_i is the symbol 1, then there exists a $j \leq n$ such that $i < j$ and $s[j,i]$ is a propositional variable.*

Observation 5.4 *If $s = s_n \dots s_1 s_0$ is a string over Σ such that s is a Polish formula, then s always starts (from the right) with a propositional variable.*

Proof: The proof is by structural induction on s. □

Definition 5.5 *Let $\sigma \in \Sigma$. Then define the **weight** of σ, denoted $weight(\sigma)$, as follows:*

$$weight(\sigma) = \begin{cases} 1 & \text{if } \sigma \text{ is } p \\ 0 & \text{if } \sigma \text{ is } 1 \text{ or } \neg \\ -1 & \text{if } \sigma \text{ is } \wedge \text{ or } \vee \end{cases}$$

The weight function can be extended to assign a weight to a string $s = s_n \dots s_0$ in Σ as follows. If $n = 0$, then $weight(s) = weight(s_0)$. Else, if $n > 0$, then $weight(s) = weight(s_n) + weight(s[n-1, 0])$.

Observation 5.6 *If $s \in \Sigma^+$ such that the length of s is n, then $weight(s) \leq n$.*

The conditions (stated in the following Lemma) that are needed for a string s over Σ to be a Polish formula, are slight modifications of those in [Bus87a]. The use of the *max* function is justified by the fact that we only want values ≥ 0 for the weight of any substring of a string $s \in \Sigma$.

Lemma 5.7 *Let $s = s_n \dots s_1 s_0$ be a string over Σ and w be $w_n \dots w_1 w_0$ such that $w_0 = max(weight(s_0), 0)$ and $w_{k+1} = max(weight(s_{k+1}) + w_k, 0)$, for k from 0 to $n-1$. Then s is a Polish formula if and only if the following conditions hold:*

1. *For every $i \leq n$, if s_i is the symbol 1, then there exists a $j \leq n$ such that $i < j$ and $s[j,i]$ is a propositional variable.*
2. *$w_n = 1$.*
3. *There exists an $i \leq n$ such that $s[i,0]$ is a propositional variable and for every $j < n$, if $j \geq i$, then $w_j > 0$.*

Proof: ($\Rightarrow$) Suppose that s is a Polish formula. We show that condition *1, 2* and *3* hold by structural induction on s. If s is a propositional variable, then those conditions hold trivially. If s is of the form $\neg\varphi$, then condition *1, 2* and *3* hold for φ, by induction hypothesis. Thus, they hold for s, since s_n is $\neg$ and $weight(\neg) = 0$. Suppose that s is of the form $\star\varphi\psi$, for $\star \in \{\wedge, \vee\}$. Assume w.l.o.g. that $\star$ is $\wedge$. Then, condition *1* holds for φ and ψ, by induction hypothesis. Clearly, condition *1* holds for $\varphi\psi$. Since s_n is $\wedge$, conditon *1* also holds for s. We next show that condition *2* holds for s. By induction hypothesis, condition *2* holds for φ and ψ. Thus, $w_{n-1} = 2$ by definition. Since $s_n = \wedge$ and $weight(\wedge) = -1$, we get that $w_n = 1$ by definition. Thus condition *2* holds for s. Finally, we show that condition *3* holds for s. Let $k \leq n$ such that $s[k,0]$ is ψ (i.e. $s[n-1,k+1]$ is φ). By induction hypothesis, condition *3* holds for $s[k,0]$. Since condition 2 holds for $s[k,0]$ (by induction hypothesis), we obtain $w_k > 0$. By induction hypothesis, condition *3* holds for $s[n-1,k+1]$. Hence, we can let $i \in \mathbb{N}$ between $k+1$ and $n-1$ such that $s[i,k+1]$ is a propositional variable. Therefore, $w_i > 0$ (since $w_k > 0$). It follows that $w_{n-2} > 0$ (since condition 3 holds for $s[n-1,k+1]$). Since condition 2 holds for $s[n-1,k+1]$, we conclude that $w_{n-1} > 0$. Thus, condition 3 holds for s.

($\Leftarrow$) Suppose that the conditions *1, 2* and *3* hold for s. We show that s is a Polish formula by induction on n. If $n = 0$, then the length of s is 1. The only string over Σ of length 1 that satisfies the conditions *1, 2* and *3* is p. Hence, s must be the symbol p. Thus, s is a Polish formula. Suppose that $n \geq 1$. By condition *1*, s_n is not the symbol 1. Therefore, we can exclude that case. Suppose that s_n is the symbol p. Since $w_n = 1$, by assumption, and $weight(s_n) = 1$, we conclude that $w_{n-1} = max(weight(s[n-1,0]), 0) = 0$. Now, condition *3* states that there exists an $i \leq n$ such that $s[i,0]$ is a propositional variable and for every $j < n$, if $j \geq i$, then $w_j > 0$. Clearly, i can't be lesser than or equal to $n-1$: w_{n-1} would then be strictly greater than 0, by condition *3*, and that implies that $w_n > 1$ (since s_n is p). Therefore, i must be equal to n. Thus, s is a propositional variable. Suppose that s_n is $\neg$. Therefore, $w_{n-1} = max(weight(s[n-1,0]), 0) = 1$. Thus, condition *2* holds for $s[n-1,0]$. Now, as condition *1* holds for s, then it also holds for $s[n-1,0]$ (since s_n is not p). As condition *3* also holds for s, it holds for $s[n-1,0]$. Therefore, $s[n-1,0]$ is a Polish formula, by induction hypothesis. By the definition of Polish formulae, s is a Polish formula. Finally, suppose that s_n is either $\wedge$ or $\vee$. Assume w.l.o.g. that it is $\wedge$. Since $weight(\wedge) = -1$, we conclude that $weight(s[n-1,0]) = 2$ (since $weight(s[n,0]) = 1$ by assumption). Thus, $w_{n-1} = 2$ by definition. Let i be the largest natural number strictly lesser than n such that condition *1, 2* and *3* hold for $s[i,0]$ (in the worst case, i concides with the "i" in condition *3*). By induction hypothesis, $s[i,0]$ is a Polish formula. We now want to show that $s[n-1,i+1]$ is also a Polish formula, i.e. it satisfies condition *1,2* and *3*. Clearly, condition *1* holds for $s[n-1,i+1]$ if it holds for s (since s_n is not p). Since $w_{n-1} = 2$ and $w_i = 1$, we get that $weight(s[n-1,i+1]) = 1$. Therefore, condition *2* holds for $s[n-1,i+1]$. We are now left with proving if condition *3* holds for $s[n-1,i+1]$, i.e. there exists a j between $n-1$ and $i+1$ such that $s[j,i+1]$ is a propositional variable and for every $k \leq n-2$, if $k \geq j$, then $w_k > 1$. We know that s_{i+1} can't be $\neg$, since we took i to be the largest natural number strictly lesser than n such that condition *1, 2* and *3* hold for $s[i,0]$. Furthermore, we know that s_{i+1} can't be $\wedge$ or $\vee$, otherwise it is a contradiction to our assumption that condition *3* holds for s. Thus, s_{i+1} is either 1 or p. Suppose that s_{i+1} is 1. Since condition *1* holds for $s[n-1, i+1]$, let j be a number between $n-1$ and $i+1$ such that $s[j, i+1]$ is a propositional variable. We next show that for every $k \leq n-2$, if $k \geq j$, then $w_k > 1$. For the sake of contradiction, suppose that there exists a k between $n-2$ and j such that $w_k \leq 1$. Let k' be the smallest number between $n-2$ and j such that $w_{k'} \leq 1$. Clearly, condition *1, 2* and *3* hold for $s[k', 0]$ and $k' > i$. Hence, a contradiction to "i is the largest number strictly lesser than n such that condition *1, 2* and *3* hold for $s[i, 0]$". Therefore, for every $k \leq n-2$, if $k \geq j$, then $w_k > 1$. For s_{i+1} is p, we apply the same reasoning as when $s[n-1, i+1]$ starts, from the right, with $p11\ldots1$. Thus, condition *3* also holds for $s[n-1, i+1]$. Since condition *1,2* and *3* hold for $s[n-1, i+1]$, $s[n-1, i+1]$ is a Polish formula by induction hypothesis. By the definition of Polish formulae, s is a Polish formula. □

Definition 5.8 *Let $s = s_n \ldots s_1 s_0$ be a string over Σ. Then define the binary string encoding $bse(s)$ of s, where bse is a mapping from Σ^+ to $\{0,1\}^*$, as follows:*

$$bse(s) \quad = \quad \begin{cases} 1000 & \textit{if } s = p \\ 1001 & \textit{if } s = 1 \\ 1010 & \textit{if } s = \neg \\ 1011 & \textit{if } s = \wedge \\ 1100 & \textit{if } s = \vee \\ bse(s[n])bse(s[n-1,0]) & \textit{if } |s| > 1 \end{cases}$$

The notation $bse(s)bse(s')$, in Definition 5.8, is understood as the concatenation of the binary string encodings of the strings s and s'. We denote by Σ_{bin} the alphabet $\{1000, 1001, 1010, 1011, 1100\}$.

Definition 5.9 *Let $\vec{x} = x_1, \ldots, x_k, \vec{X} = X_1, \ldots X_l$, $\vec{n} = n_1, \ldots, n_k \in \mathbb{N}$, $\vec{N} = N_1, \ldots, N_l \in \mathcal{P}_{fin}(\mathbb{N})$. A relation $R \subseteq \mathbb{N}^k \times \mathcal{P}_{fin}(\mathbb{N})^l$ is Σ_0^B-**definable** if there exists a Σ_0^B-formula $\varphi(\vec{x}, \vec{X})$ such that for all $(\vec{n}, \vec{N}) \in \mathbb{N}^k \times \mathcal{P}_{fin}(\mathbb{N})^l$,*

$$(\vec{n}, \vec{N}) \in R \textit{ iff } \underline{\mathbb{N}}_2 \models \varphi(\underline{\vec{n}}, \vec{X})[\alpha(\vec{N}/\vec{X})].$$

*We say that $\varphi(\vec{x}, \vec{X})$ Σ_0^B-**defines** R.*

Remember that the goal of this subsubsection is to Σ_0^B-define a formula Fla which defines the relation $Fla(X, W)$ which holds if and only if X encodes a Polish formula $s = s_n \ldots s_1 s_0$, where $s_i \in \Sigma$, and W encodes $w = w_n \ldots w_1 w_0$, where w is defined as in Lemma 5.7 but this time relative to s. Figure 1 provides a high-level description of the structures of X and W, where X and W encode s and w respectively. Before commenting on Figure 1, we first introduce the following abbreviations.

Notation

- $W_i[j]$ is a shorthand for $W[(n+2) \cdot i + j]$.
- $X_i[j]$ is a shorthand for $X[4i + j]$.

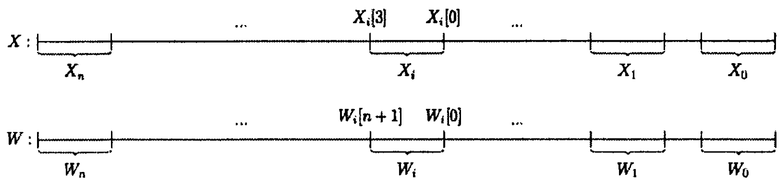

Figure 1: The structures of X and W.

- $W_i[j,k]$ is a shorthand for $W[(n+2)\cdot i+j,(n+2)\cdot i+k]$, where $j \geq k$.
- $X_i[j,k]$ is a shorthand for $X[4i+j, 4i+k]$, where $j \geq k$.
- W_i is a shorthand for $W_i[n+1, 0]$.
- X_i is a shorthand for $X_i[3,0]$.
- $X_{i\to j}$ is a shorthand for $X_i X_{i-1} \dots X_j$.
- We write "W_i encodes j" if and only if for every $k < j$, $W_i[k] = 1$, and for every $k \leq n$, if $k \geq j$, then $W_i[k] = 0$.

The first axis, labelled "X :" in Figure 1, represents X as a binary string. As we see from Figure 1, X is divided into $n+1$ blocks. A block X_i is viewed as the binary string encoding of s_i in s.

The second axis, labelled "W :" in Figure 1, represents W as a binary string. As with the first axis, W is divided into $n+1$ blocks. Each block W_i has length $n+2$ and where $W_i[n+1]$ is always 1. $W_i[n,0]$ is then the representation of w_i (in W) in unary. That is to say, if $w_i = j$, then W_i encodes j.

Lemma 5.10 *Let X be $X_n \dots X_1 X_0$, where $X_i \in \Sigma_{bin}$ for every i from 0 to n, and W be $W_n \dots W_1 W_0$ such that:*

(W1). $|W_i| = (n+2)$.

(W2). $W_i[n+1] = 1$, *for every i from 0 to n.*

(W3). *If X_0 encodes p, then W_0 encodes 1. Else if X_0 encodes $1, \neg, \wedge$ or $\vee$, then W_0 encodes 0.*

(W4.) *For every i from 1 to n, we have that:*

  (W4.1). *If X_i encodes 1 or $\neg$, then $W_i = W_{i-1}$.*

  (W4.2). *If X_i encodes p, then there exists a j between 1 and $n+1$ such that W_i encodes j and W_{i-1} encodes $j-1$.*

  (W4.3). *If X_i encodes either $\wedge$ or $\vee$, then there exists a j between 0 and n such that W_{i-1} encodes j and W_i encodes $j \dot{-} 1$.*

Then X encodes a Polish formula if and only if the following conditions hold:

(X1). *For every $i \leq n$, if X_i encodes 1, then there exists a $j \leq n$ such that $i < j$ and $X_{j \to i}$ encodes a propositional variable.*

(X2). *W_n encodes 1.*

(X3). *There exists an $i \leq n$ such that $X_{i \to 0}$ encodes a propositional variable and for every $j < n$, if $j \geq i$, then W_j encodes k, where $k > 0$.*

Note that Lemma 5.10 is a natural translation of Lemma 5.7.

In the following, we often use the same notation for both the Σ_0^B-formula and the relation that it defines. Also, remember that we identify a finite non-empty subset of $\mathbb{N}$ with its binary string representation. Finally, $b_1(x,y)$ is an abbreviation for $4x+y$ and $b_2(x,y)$ is an abbreviation for $(\underline{n}+2)x+y$, where n is the length of the Polish formula under consideration.

We are now going to provide three examples that illustrate how to Σ_0^B-formulate the conditions in Lemma 5.10, that X and W must satisfy to encode a Polish formula. First, note that two of the conditions that (X, W) must satisfy in order to be an encoding of a Polish formula is that $|X| = 4(n+1)$, for some $n \geq 0$, and that every block X_i of X is an encoding of a symbol in Σ. Also, note that one of the conditions that W must satisfy is that $|W| = (n+1)(n+2)$. In the following examples, assume that (X, W) satisfy the two conditions mentioned in this paragraph.

For the first example, we Σ_0^B-formulate "If X_0 encodes p, then W_0 encodes 1. Else if X_0 encodes $1, \neg, \wedge$ or $\vee$, then W_0 encodes 0." as follows:

$$\begin{gathered}(XEnc_p(X,0) \to WEnc_1(W,0)) \\ \wedge \\ \left(\begin{gathered}(XEnc_1(X,0) \vee XEnc_\neg(X,0) \vee XEnc_\wedge(X,0) \vee XEnc_\vee(X,0)) \\ \to \\ WEnc_0(W,0)\end{gathered}\right),\end{gathered}$$

where

(11)

$$\begin{aligned}
XEnc_p(X,x) &=_{df} X(b_1(x,3)) \wedge \neg X(b_1(x,2)) \wedge \neg X(b_1(x,1)) \wedge \neg X(b_1(x,0))) \\
XEnc_1(X,x) &=_{df} X(b_1(x,3)) \wedge \neg X(b_1(x,2)) \wedge \neg X(b_1(x,1)) \wedge X(b_1(x,0))) \\
XEnc_\neg(X,x) &=_{df} X(b_1(x,3)) \wedge \neg X(b_1(x,2)) \wedge X(b_1(x,1)) \wedge \neg X(b_1(x,0)) \\
XEnc_\wedge(X,x) &=_{df} X(b_1(x,3)) \wedge \neg X(b_1(x,2)) \wedge X(b_1(x,1)) \wedge X(b_1(x,0)) \\
XEnc_\vee(X,x) &=_{df} X(b_1(x,3)) \wedge X(b_1(x,2)) \wedge \neg X(b_1(x,1)) \wedge \neg X(b_1(x,0))
\end{aligned}$$

and

$$\text{(12)} \qquad WEnc_m(W,x) =_{df} \left(\begin{gathered}(\forall y < \underline{m})(W(b_2(x,y))) \\ \wedge \\ (\forall y \leq \underline{n})(y \geq \underline{m} \to \neg W(b_2(x,y)))\end{gathered}\right).$$

Here, $XEnc_\sigma(X,x)$, where $\sigma \in \Sigma$, asserts that X_x encodes the symbol σ. $WEnc_m(W,x)$ asserts that W_x encodes m.

Before we go to the next example, let us first Σ_0^B-define the function $x \dot{-} y = max(0, x - y)$ as follows:

$$z = x \dot{-} y =_{df} ((y + z = x) \vee (x \leq y \wedge z = 0)) \tag{13}$$

For the second example, we Σ_0^B-formulate "For every i from 1 to n, if X_i encodes 1 or $\neg$, then $W_i = W_{i-1}$." as follows:

$$\begin{aligned}&(\forall x \leq \underline{n})(x \geq 1 \rightarrow \\ &\quad ((XEnc_1(X, x) \vee XEnc_\neg(X, x)) \rightarrow \\ &\qquad (\exists y \leq x)(y = x \dot{-} 1 \wedge Eq(W, x, y)))),\end{aligned}$$

where $Eq(W, x, y)$ is defined by:

$$(\forall z \leq \underline{n} + 1)(W(b_2(x, z)) \leftrightarrow W(b_2(y, z))) \tag{14}$$

For the last example, we Σ_0^B-formulate "For every $i \leq n$, if X_i encodes 1, then there exists a $j \leq n$ such that $i < j$ and $X_{j \to i}$ encodes a propositional variable." as follows:

$$(\forall x \leq \underline{n})(XEnc_1(X, x) \rightarrow (\exists y \leq \underline{n})((x < y) \wedge XEnc_{var}(X, x, y)))$$

where $XEnc_{var}(X, x, y)$, which asserts that $X_{x \to y}$ encodes a propositional variable $p11\ldots1$ with $(x - y)$ many 1's, is defined by:

(15)

$$(x \geq y) \wedge (\exists z \leq x)(x = z + y \wedge XEnc_p(X, x) \wedge (\forall z_0 < z)(XEnc_1(X, z_0)))$$

The other conditions that X and W must satisfy, for (X, W) to encode a Polish formula, can be Σ_0^B-formulated in the same way as those examples. Hence, let $Fla(X, W)$ be a conjunction over the conditions (W1), (W2), (W3), (W4), (X1), (X2), (X3) and Σ_0^B-defines the relation $Fla(X, W)$.

5.1.2 Encoding Truth Assignments

Our way of encoding a truth assignment to the variables in a Polish formula follows [CN10].

Recall that $|S|$ is the length of the binary string representation of the set S (finite subset of $\mathbb{N}$) and that the indices of propositional variables in a Polish formula are written in unary notation. Hence, if s is a string in Σ^* such that s is a Polish formula, then there are $\leq |s|$ distinct variables in s and their indices are $\leq |s|$. Now, suppose that X encodes s. Then, a set $Z \subseteq \mathbb{N}$ specifies a truth assignment to the variables $p11\ldots1$ in X as follows:

$$p11\ldots1 \text{ is assigned the value of } Z(|p11\ldots1| - 1).$$

Therefore, all truth assignments to the variables in X can be specified by sets $Z \subseteq \mathbb{N}$ such that $|Z| \leq |X|$. Thus, the Σ_0^B-formula $Assign(X, W, Z)$, where the relation $Assign(X, W, Z)$ holds if and only if the relation $Fla(X, W)$ holds and Z specifies a truth assignment to the variables in X, is defined by:

$$Fla(X, W) \wedge (|Z| \leq |X|) \tag{16}$$

We will next define the Σ_0^B-formula $Eval(X, W, Z, Z')$, where the relation $Eval(X, W, Z, Z')$ holds if and only if the relation $Assign(X, W, Z)$ holds and Z' extends Z to the subformulae of the formulae encoded by X. For that, we first need to define what it means for $X_{i \to j}$ to encode a subformula of a Polish formula encoded by X and define how Z' extends Z.

Lemma 5.11 *Let $s = s_n \ldots s_1 s_0$ be a Polish formula and $w = w_n \ldots w_1 w_0$ be defined as in Lemma 5.7. Then, for every $j, k \in \mathbb{N}$ such that $n \geq j \geq k \geq 0$, $s_j s_{j-1} \ldots s_k$ is a Polish formula if and only if the following conditions hold:*

1. *For every $i \leq j$, if $i \geq k$, then the following statement must hold. If s_i is the symbol 1, then there exists an $l \leq j$ such that $i < l$ and $s[l, i]$ is a propositional variable.*
2. *There exists an $i \leq j$ such that $i \geq k$ and $s[i, k]$ is a propositional variable and for every $l < j$, if $l \geq i$, then $w_l \geq w_i$, and $w_j = w_i$.*

Proof: The proof is similar to Lemma 5.7. □

Lemma 5.12 *Let $X = X_n \ldots X_1 X_0$ encode a Polish formula, where $X_i \in \Sigma_{bin}$ for every i from 0 to n, and W satisfies the conditions* (W1), (W2), (W3) *and* (W4) *in Lemma 5.10. Then, for every $j, k \in \mathbb{N}$ such that $n \geq j \geq k \geq 0$, $X_{j \to k}$ encodes a Polish formula if and only if the following conditions hold:*

1. *For every $i \leq j$, if $i \geq k$, then the following statement must hold. If X_i encodes 1, then there exists an $l \leq j$ such that $i < l$ and $X_{l \to i}$ is a propositional variable.*
2. *There exists an $i \leq j$ such that $i \geq k$ and $X_{i \to k}$ is a propositional variable and for every $l < j$, if $l \geq i$ and W_l encodes n_0 and W_i encodes n_1, then $n_0 \geq n_1$, and $W_j = W_i$.*

Clearly, Lemma 5.12 is a natural translation of Lemma 5.11.

Lemma 5.13 *Let $X = X_n \ldots X_1 X_0$ encode a Polish formula, where $X_i \in \Sigma_{bin}$, and W be defined as in Lemma 5.10. Then, for every $j, k \in \mathbb{N}$ such that $n \geq j \geq k \geq 0$, $X_{j \to k}$ encodes a subformula of the Polish formula encoded by X if and only if the following conditions hold:*

(S1). *$X_{j \to k}$ encodes a Polish formula.*

(S2). *If $k > 0$, then X_{k-1} doesn't encode the symbol 1.*

Proof: Obvious. □

Condition *1* and *2*, in Lemma 5.13, can be expressed by a Σ_0^B-formula. Thus, let $Subf(X, x, y, W)$ be a Σ_0^B-formula (a conjunction of (S1), (S2), (W1), (W2), (W3), (W4)) which asserts that $(X_{x \to y}, W)$ encodes a subformula of the formula encoded by X.

Assume that $Assign(X, W, Z)$ holds. A set $Z' \subseteq \mathbb{N}$ extends Z to the subformulae of X as follows:

1. For every $j, k \leq |X|$, if $X_{j \to k}$ encodes a propositional variable and that $Subf(X, j, k, W)$ holds, then $Z'(j)$ holds if and only if $Z(j-k)$ holds.

2. For every $j, k \leq |X|$, if the relation $Subf(X, j, k, W)$ holds and X_j encodes $\neg$, then $Z'(j)$ holds if and only if $\neg Z'(j-1)$ holds.

3. For every $j, k \leq |X|$, if $Subf(X, j, k, W)$ and X_j encodes $\star \in \{\wedge, \vee\}$, then there exists an l such that: $k \leq l < j-1$, $Subf(X, j-1, l+1, W)$ and $Subf(X, l, k, W)$ hold and $Z'(j)$ holds if and only if $Z'(j-1) \star Z'(l)$.

4. $|Z'| \leq |X|$.

Clearly, all those four conditions are Σ_0^B-definable. Let $\varphi_{C_1}, \varphi_{C_2}, \varphi_{C_3}$ and φ_{C_4} be the Σ_0^B-formulae that express condition 1, 2, 3 and 4 above, respectively. Then,

$$Eval(X, W, Z, Z') =_{df} Assign(X, W, Z) \wedge \varphi_{C_1} \wedge \ldots \wedge \varphi_{C_4}. \tag{17}$$

5.1.3 Encoding Polytime Turing Machine Computations

From now on, we assume that every Turing machine M that will be discussed is a polytime Turing machine which takes binary strings (empty string included) as inputs and outputs binary strings (empty string included). Furthermore, for a Turing machine $M = (K, \Sigma, \delta, s)$, we assume that $\Sigma = \{0, 1, 2, 3\}$, where 2 and 3 always encode $\triangleright$ and the blank symbol, respectively ($\triangleright$ and $\sqcup$ will often be used to refer to 2 and 3, respectively), and $K = \{4, 5, \ldots, |\Sigma| + |K| - 1\}$, where 4 and 5 always encode s and h, respectively (s and h will often be used to refer to 4 and 5, respectively). This idea of coding symbols and states of a Turing machine into natural numbers is from [Pap94]. Additionally, Turing machines will never write a $\triangleright$ on their string except when they see one. Moreover, a Turing machine configuration (q, w, u), as defined in Definition 2.2, is redefined here as wqu. Here, $w = w_n w_{n-1} \ldots w_1 w_0$ such that $w_n = \triangleright, w_0$ is the symbol read by M at state q and if $n \geq 1$, then for every $i \leq n-1$, $w_i \in \Sigma \backslash \{\triangleright\}$ (since we never write $\triangleright$ except when we see one); $q \in K$ and $u \in (\Sigma \backslash \{\triangleright\})^*$. Finally, if $w_n w_{n-1} \ldots w_0 h u_{m-1} \ldots u_0$ is the final configuration of a Turing machine M on input X, then $M(X) = w_{n-1} \ldots w_0$, where $w_i \in \{0, 1\}$ for every i from 0 to $n-1$.

5.1.3.1 A method of encoding configurations of a Turing machine on a given input. The purpose of this paragraph is to describe a method of encoding configurations of a Turing machine on a given input.

Definition 5.14 *Let $M = (K, \Sigma, \delta, s)$ be a Turing machine. Then, we define enc to be a mapping from $\Sigma \cup K$ to $\{0,1\}^*$ such that for all $n \in \Sigma \cup K$,*

$$enc(n) = 1b \tag{18}$$

where b is the binary representation of n such that $|b| = \lceil log_2(|\Sigma| + |K|) \rceil$.

enc can then be extended to assign a binary string to a string $\mathrm{s} \in \Sigma^+$ as follows:

$$enc(\mathrm{s}) = \begin{cases} 1b, \text{ where } b \text{ is defined as in } (18) & \text{if } \mathrm{s} \in \Sigma \\ enc(\mathrm{s}[n])enc(\mathrm{s}[n-1, 0]) & \text{if } |\mathrm{s}| > 1 \end{cases}$$

where enc(s)...*enc*(s′) *is the result of concatenating enc*(s) *and enc*(s′).

For example, if $|\Sigma| + |K| = 8$, then $\lceil log_2(|\Sigma| + |K|)\rceil = 3$. Hence, $enc(3) =$ 1011 and $enc(33) = 10111011$.

This is a preparation for the definition of the encoding of a configuration of a Turing machine on a given input. Let $t(|X|)$ be a bound on the running time of a Turing machine M on input X. Hence, for any configuration wqu of M on X, $|w| + |u| \leq t(|X|) + 1$. Thus, $|enc(w)enc(q)enc(u)enc(\sqcup)| \leq k \cdot (t(|X|) + 3)$, where $k = 1 + \lceil log_2(|\Sigma| + |K|)\rceil$ (here, k is the length of the encoding of a symbol $\sigma \in (\Sigma \cup K)$).

Notation Let $M = (K, \Sigma, \delta, s)$ be a Turing machine. From now on, let $1 + \lceil log_2(|\Sigma| + |K|)\rceil$ be denoted by k_M. Additionally, $t_M(|X|)$ always denotes the bound on the running time of a Turing machine M on input X. At the formal language level, k_M denotes the numeral that evaluates to $1 + \lceil log_2(|\Sigma| + |K|)\rceil$ in the standard model and $t_M(|X|)$ denotes a number term that evaluates to the bound on the running time of M on X in the standard model. We abbreviate $t_M(|X|)$ by t_M.

Definition 5.15 *We define* ***the encoding of a configuration*** wqu ***of a Turing machine*** M ***on input*** X *to be the binary string*

$$V = enc(w)enc(q)enc(u)enc(\sqcup)enc(\sqcup \ldots \sqcup) \tag{19}$$

such that $|V| = k_M(t_M + 3)$.

Note that the substring $enc(\sqcup \ldots \sqcup)$ of V, in (19), maybe an empty string and (19) always ends with a $\sqcup$ (that will be clear later, when we describe how to recognise if two binary strings encode two consecutive configurations of a certain Turing machine on a certain input). Let $b_3(x, y)$ be a shortand for $k_M \cdot x + y$. The other conditions are clearly Σ_0^B-formulable. Thus, let $Conf_M(V, X)$ be a Σ_0^B-formula which asserts that V is a potential encoding of a configuration of a Turing machine M on input X.

5.1.3.2 Σ_0^B-defining the relation $Init_M(X, V)$.

Remember that the initial configuration of a Turing machine M on input X is

$$\triangleright s X(|X| - 1) \ldots X(1)X(0)$$

and whose encoding is the binary string

$$enc(\triangleright)enc(s)enc(X)enc(\sqcup)enc(\sqcup \ldots \sqcup)$$

of length $k_M(t_M + 3)$. Thus, the relation $Init_M(X, V)$, which holds if and only if $V = V_{n+1}V_n \ldots V_0$ encodes the initial configuration of M on input X, is Σ_0^B-defined as follows:

(20)

$$Init_M(X,V)$$

$$=_{df}$$

$$|V| = k_M(t_M+3)$$

$$\wedge$$

$$Symbol^M_{\triangleright}(V, t_M+2) \wedge State^M_s(V, t_M+1)$$

$$\wedge$$

$$(\forall x < |X|)(\exists y \leq |X|)(\exists y_0 \leq y)(\exists z_0 \leq t_M)\begin{pmatrix} y = |X| \dot{-} 1 \\ \wedge \\ y_0 = y \dot{-} x \\ \wedge \\ z_0 = t_M \dot{-} x \\ \wedge \\ (X(y_0) \leftrightarrow Symbol^M_1(V, z_0)) \\ \wedge \\ (\neg X(y_0) \leftrightarrow Symbol^M_0(V, z_0)) \end{pmatrix}$$

$$\wedge$$

$$(\exists y \leq t_M+1)(y = (t_M+1) \dot{-} |X| \wedge (\forall x < y)(Symbol^M_{\sqcup}(V,x)))$$

where $Symbol^M_0(V,x)$, $Symbol^M_1(V,x)$, $Symbol^M_{\triangleright}(V,x)$, $Symbol^M_{\sqcup}(V,x)$ and $State^M_s(V,x)$ are defined as follows:

$$(21) \qquad State^M_s(V,x) =_{df} (\exists z < k_M)\begin{pmatrix} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x,z)) \\ \wedge \\ V(b_3(x,2)) \\ \wedge \\ (\forall y < z)(y > 2 \rightarrow \neg V(b_3(x,y))) \\ \wedge \\ (\forall y < 2)(\neg V(b_3(x,y))) \end{pmatrix},$$

which means that $s = 4 = 100_2$ is encoded by $10\ldots0100$, and

$$Symbol_0^M(V,x) =_{df} (\exists z < k_M)\begin{pmatrix} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x,z)) \\ \wedge \\ (\forall y < z)(\neg V(b_3(x,y))) \end{pmatrix}$$

$$Symbol_1^M(V,x) =_{df} (\exists z < k_M)\begin{pmatrix} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x,z)) \\ \wedge \\ V(b_3(x,0)) \\ \wedge \\ (\forall y < z)(y > 0 \rightarrow \neg V(b_3(x,y))) \end{pmatrix}$$

(22)

$$Symbol_{\triangleright}^M(V,x) =_{df} (\exists z < k_M)\begin{pmatrix} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x,z)) \\ \wedge \\ V(b_3(x,1)) \\ \wedge \\ \neg V(b_3(x,0)) \wedge \\ (\forall y < z)(y > 1 \rightarrow \neg V(b_3(x,y))) \end{pmatrix}$$

$$Symbol_{\sqcup}^M(V,x) =_{df} (\exists z < k_M)\begin{pmatrix} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x,z)) \\ \wedge \\ V(b_3(x,0)) \\ \wedge \\ V(b_3(x,1)) \\ \wedge \\ (\forall y < z)(y > 1 \rightarrow \neg V(b_3(x,y))) \end{pmatrix}$$

where $b_3(x,y)$ is a shortand for $k_M \cdot x + y$.

5.1.3.3 Σ_0^B-defining the relation $Yields_M(V,V',X)$. In this paragraph, we Σ_0^B-define the relation $Yields_M(V,V',X)$, which holds if and only if V and V' encode two consecutive configurations of the Turing machine M on input X. Before we do so, first consider the following example.

Example 5.16 *Consider the Turing machine $M = (K,\Sigma,\delta,s)$, where $\Sigma = \{0,1,\triangleright,\sqcup\}$ and $K = \{s,h,6\}$ and δ as shown in Table 1. M simply turns its input X into a string of 0's if X is not the emptystring; if X is the empty string, then $M(\epsilon) = \epsilon$, where ϵ denotes the empty string. Note that we omit rules that will never be encountered in a legal computation and, since $|\Sigma| + |K| = 7$, the length of the binary string encoding of every symbol in $(\Sigma \cup K)$ is then 4.*

Let us consider the two configurations $c_0 = \triangleright s11$ and $c_1 = \triangleright 1s1$ of M on

$q \in K$	$\sigma \in \Sigma$	$\delta(q, \sigma)$
s	0	$(s, 0, \rightarrow)$
s	1	$(s, 0, \rightarrow)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	$\sqcup$	$(6, \sqcup, \leftarrow)$
6	0	$(h, 0, -)$
6	$\triangleright$	$(h, \triangleright, -)$

Table 1: A Turing machine.

input 11, *and whose binary string encodings are* c'_0 *and* c'_1 *respectively, where*

$$\begin{aligned} c'_0 = &\quad enc(\triangleright)enc(s)enc(1)enc(1)enc(\sqcup)enc(\sqcup \ldots \sqcup) \\ c'_1 = &\quad enc(\triangleright)enc(1)enc(s)enc(1)enc(\sqcup)enc(\sqcup \ldots \sqcup). \end{aligned}$$

Now, a way to tell, if c'_0 *and* c'_1 *encode two consecutive configurations of* M *on* 11, *is that they are identical except that the substring* $enc(\triangleright)enc(s)enc(1)$ *of* c'_0 *has been replaced by the substring* $enc(\triangleright)enc(1)enc(s)$ *of* c'_1 *and this replacement corresponds to the rule* $\delta(s, \triangleright) = (s, \triangleright, \rightarrow)$, *in Table 1 (this idea is from [Pap94]). Thus, a move of* M *entails a replacement of triples of binary strings. The complete table of these triples and their replacements, for the machine* M *of Table 1, is shown in Table 2. Table 2 is then encoded into a binary string* T

Original substring	Replacement
$lc_{0,0} = enc(\triangleright)enc(s)enc(0)$	$lc_{0,1} = enc(\triangleright)enc(0)enc(s)$
$lc_{1,0} = enc(\triangleright)enc(s)enc(1)$	$lc_{1,1} = enc(\triangleright)enc(1)enc(s)$
$lc_{2,0} = enc(\triangleright)enc(s)enc(\sqcup)$	$lc_{2,1} = enc(\triangleright)enc(\sqcup)enc(s)$
$lc_{3,0} = enc(0)enc(s)enc(0)$	$lc_{3,1} = enc(0)enc(0)enc(s)$
$lc_{4,0} = enc(0)enc(s)enc(1)$	$lc_{4,1} = enc(0)enc(1)enc(s)$
$lc_{5,0} = enc(0)enc(s)enc(\sqcup)$	$lc_{5,1} = enc(0)enc(\sqcup)enc(s)$
$lc_{6,0} = enc(1)enc(s)enc(0)$	$lc_{6,1} = enc(0)enc(0)enc(s)$
$lc_{7,0} = enc(1)enc(s)enc(1)$	$lc_{7,1} = enc(0)enc(1)enc(s)$
$lc_{8,0} = enc(1)enc(s)enc(\sqcup)$	$lc_{8,1} = enc(0)enc(\sqcup)enc(s)$
$lc_{9,0} = enc(\sqcup)enc(s)enc(0)$	$lc_{9,1} = enc(6)enc(\sqcup)enc(0)$
$lc_{10,0} = enc(\sqcup)enc(s)enc(1)$	$lc_{10,1} = enc(6)enc(\sqcup)enc(1)$
$lc_{11,0} = enc(\sqcup)enc(s)enc(\sqcup)$	$lc_{11,1} = enc(6)enc(\sqcup)enc(\sqcup)$
$lc_{12,0} = enc(0)enc(6)enc(0)$	$lc_{12,1} = enc(0)enc(h)enc(0)$
$lc_{13,0} = enc(0)enc(6)enc(1)$	$lc_{13,1} = enc(0)enc(h)enc(1)$
$lc_{14,0} = enc(0)enc(6)enc(\sqcup)$	$lc_{14,1} = enc(0)enc(h)enc(\sqcup)$
$lc_{15,0} = enc(\triangleright)enc(6)enc(0)$	$lc_{15,1} = enc(\triangleright)enc(h)enc(0)$
$lc_{16,0} = enc(\triangleright)enc(6)enc(1)$	$lc_{16,1} = enc(\triangleright)enc(h)enc(1)$
$lc_{17,0} = enc(\triangleright)enc(6)enc(\sqcup)$	$lc_{17,1} = enc(\triangleright)enc(h)enc(\sqcup)$

Table 2: A Table of triples and their replacements.

such that $|T| = 6k_M \times 3 \cdot r$, *where* r *is the number of rows in Table 1, and for every* $i \leq r - 1$, *there exists a* $j \leq r - 1$ *such that* $V_i = lc_{j,0}lc_{j,1}$, *where* $V_i = V[6k_M \cdot i + (6k_M - 1), 0]$. *An example of a valid encoding of Table 2 is as follows:*

(23) $$lc_{1,0}lc_{1,1}lc_{2,0}lc_{2,1} \ldots lc_{12,0}lc_{12,1}$$

In general, for a Turing machine M, its table of triples and their replacements is encoded into a binary string T_M satisfying the same conditons as T above, but this time relative to T_M.

We are now ready to Σ_0^B-define the relation $Yields_M(V, V', X)$. For that, let $T \in \mathcal{P}_{fin}(\mathbb{N})$ such that T is an encoding of the table of triples and their replacements for M. Then,

(24)
$$
\begin{gathered}
Yields_M(V, V', X) \\
=_{df} \\
|V| = k_M(t_M + 3) \wedge |V'| = |V| \\
\wedge \\
(\exists x \le t_M + 1) \left(\begin{gathered} Replacement_M(V, V', x) \\ \wedge \\ (\exists z \le x)(z = x \dot{-} 1 \wedge (\forall y < k_M \cdot z)(V(y) \leftrightarrow V'(y))) \\ \wedge \\ (\forall y < k_M(t_M + 3) \\ (y \ge k_M(x+2) \rightarrow (V(y) \leftrightarrow V'(y))) \end{gathered} \right)
\end{gathered}
$$

where $Replacement_M(V, V', x)$ asserts that the substring $V'_{x+1}V'_xV'_{x-1}$ of V' is a valid replacement of the substring $V_{x+1}V_xV_{x-1}$ of V (i.e. there exists two substrings T_{y+1} and T_y of T such that T_y is a replacement of T_{y+1}, in the table of triples and their replacements for M, and $T_{y+1} = V_{x+1}V_xV_{x-1}$ and $T_y = V'_{x+1}V'_xV'_{x-1}$) and is defined as follows

(25)
$$
\begin{gathered}
Replacement_M(V, V', x) \\
=_{df} \\
(\exists z_0 \le x) \left(\begin{gathered} z_0 = x \dot{-} 1 \\ \wedge \\ (\exists y \le |T|)(\forall z < 3k_M) \begin{pmatrix} (T(3k_M(2y+1) + z) \leftrightarrow V(k_M \cdot z_0 + z)) \\ \wedge \\ (T(3k_M \cdot 2y + z) \leftrightarrow V'(k_M \cdot z_0 + z)) \end{pmatrix} \end{gathered} \right)
\end{gathered}
$$

Figure 2 shows a pictorial description of (25).

5.1.3.4 Σ_0^B-defining the relation $Out_M(V, X, Y)$. Remember that for a final configuration whu, where $w = w_n w_{n-1} \ldots w_0$, of a Turing machine M on input X, we have that $M(X) = w_{n-1} \ldots w_0$, where $w_i \in \{0, 1\}$ for every i from 0 to $n-1$.

The relation $Out_M(V, X, Y)$, which holds iff V encodes the final configuration of M on input X and $M(X) = Y$, is Σ_0^B-defined as follows:

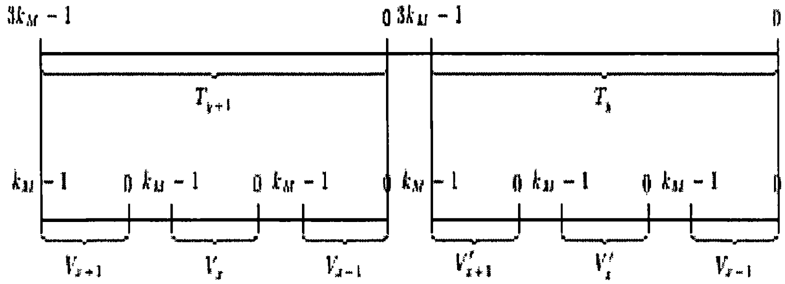

Figure 2: Pictorial description of what the formula $Replacement_M$ expresses.

$$Out_M(V, X, Y)$$

$$=_{df}$$

$$|V| = k_M(t_M + 3)$$

$$\wedge$$

(26)
$$(\exists x \leq t_M + 2)\left(\begin{array}{c} x = (t_M + 2) \dot{-} |Y| \\ \wedge \\ (\forall y < |Y|)\left(\begin{array}{c} (Y(y) \leftrightarrow Symbol_1^M(V, x + y)) \\ \wedge \\ (\neg Y(y) \leftrightarrow Symbol_0^M(V, x + y)) \end{array}\right) \end{array}\right)$$

$$\wedge$$

$$(\exists x \leq t_M + 1)(x = (t_M + 1) \dot{-} |Y| \wedge State_h^M(V, x))$$

where $State_h^M(V, x)$ is defined as follows:

(27)
$$(\exists z < k_M)\left(\begin{array}{c} z = k_M \dot{-} 1 \\ \wedge \\ V(b_3(x, z)) \\ \wedge \\ V(b_3(x, 0)) \\ \wedge \\ \neg V(b_3(x, 1)) \\ \wedge \\ V(b_3(x, 2)) \\ \wedge \\ (\forall y < z)(y > 2 \rightarrow \neg V(b_3(x, y))) \end{array}\right)$$

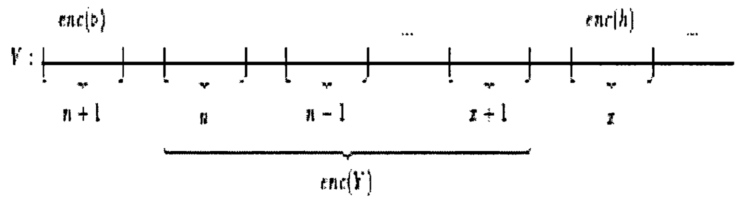

Figure 3: A pictorial description of (26). Here, $n = t_M + 1$.

Figure 3 shows a pictorial description of (26).

5.1.3.5 Σ_0^B-define the relation $Comp_g(V, Y, X)$. The goal of this paragraph is to Σ_0^B-define the relation $Comp_g(V, Y, X)$, which holds if and only if V encodes the computation of a Turing machine and V shows that $g(Y) = X$, where g is a polytime computable onto function from $\{0,1\}^*$ to $\{0,1\}^*$. To Σ_0^B-define $Comp_g(V, Y, X)$, we first need to describe a method of encoding the computation of a Turing machine M on X. For that, the following definitions are needed, but first recall that when writing $Z(x, y)$, we mean $Z(\langle x, y\rangle)$, where $\langle x, y\rangle$ is the pairing function.

Definition 5.17 *[CN10] The function $Z^{[x]}$ is defined by*

(28) $$Z^{[x]}(i) \leftrightarrow (i < |Z| \wedge Z(x, i))$$

Definition 5.18 *[CN10] The* ***string tupling function*** *$\langle X_0, X_1, \ldots, X_{n-1}\rangle$ is defined by*

(29) $$\langle X_0, \ldots, X_{n-1}\rangle(i) \leftrightarrow (i = \langle j, x\rangle) \wedge (j < n) \wedge X_j(x)$$

Definition 5.19 *Let $\psi(U)$ be a Σ_0^B-formula. Then we denote by $\psi(V^{[x]})$ the result of replacing every occurence of $U(t)$ in ψ by $V^{[x]}(t')$, where $t = \langle x, t'\rangle$.*

The computation of a Turing machine M on an input X can be encoded by a binary string $V = \langle V^{[0]}, V^{[1]} \ldots V^{[t_M]}\rangle$ such that $V^{[0]}$ is the initial configuration of M on X and for every $i < t_M$, $Yields_M(V^{[i]}, V^{[i+1]}, X)$ holds, and $Out_M(V^{[t_M]}, X, M(X))$ holds. Note that the length of V is bounded by $\langle t_M, k_M(t_M + 2)\rangle$.

Now, let g be a polytime computable onto function from $\{0,1\}^*$ to $\{0,1\}^*$ and M be a polytime Turing machine that computes g. Then, we Σ_0^B-define the relation $Comp_g(V, Y, X)$ as follows:

(30) $$\begin{aligned} Comp_g(V, Y, X) =_{df}\ & |V| \leq \langle t_M, k_M(t_M + 2)\rangle \wedge \\ & \varphi_M(X, V) \wedge Out_M(V^{[t_M]}, X, Y) \end{aligned}$$

where $\varphi_M(X, V)$ is defined as follows:

(31) $$\varphi_M(X, V) =_{df} Init_M(X, V^{[0]}) \wedge (\forall x < t_M)(Yields_M(V^{[x]}, V^{[x+1]}, X))$$

5.1.4 Σ_0^B-formulation of the Reflection Principle

From now on, we assume that propositional proof systems are defined as in [CR79], but whose domains are $\{0,1\}^*$, i.e. propositional proof systems are polytime computable onto functions from $\{0,1\}^*$ to TAUT, where TAUT $\subset \{0,1\}^*$.

We finally come to the central point of this subsection, which is to Σ_0^B-formulate the reflection principle for a propositional proof system g which states that

(32)
$$\forall X(\exists Y(g(Y) = X) \Rightarrow X \in \text{TAUT})$$

We Σ_0^B-formulate the reflection principle of a proof system g as in [Coo06]:

(33)
$$\mathit{Sound}_g(X, W, Z, Z', V, Y)$$

$$=_{df}$$

$$\mathit{Eval}(X, W, Z, Z') \wedge \mathit{Comp}_g(V, Y, X) \rightarrow (\exists x \leq |X|)(|X| = 4(x+1) \wedge Z'(x))$$

where $Z'(x)$ is the truth value of the entire Polish formula encoded by X (remember that a symbol in $\{p, 1, \neg, \wedge, \vee\}$ is encoded by a binary string of length 4).

Theorem 5.20 *Let g be a polytime computable onto function from $\{0,1\}^*$ to $\{0,1\}^*$. Then $\underline{\mathbb{N}}_2 \models \forall \mathit{Sound}_g$ if and only if $\forall X(\exists Y(g(Y) = X) \Rightarrow X \in \text{TAUT})$.*

Proof: ($\Rightarrow$) Suppose that $\forall \mathit{Sound}_g$ is true in the standard model. We show that $\forall X(\exists Y(g(Y) = X) \Rightarrow X \in \text{TAUT})$. For the sake of contradiction, suppose that $\exists X(\exists Y(g(Y) = X) \wedge X \notin \text{TAUT})$. Let $X, W, Z, Z' \in \{0,1\}^*$ such that $\mathit{Eval}(X, W, Z, Z')$ is true in $\underline{\mathbb{N}}_2$ and let $V \in \{0,1\}^*$ such that $\mathit{Comp}_g(V, Y, X)$ is true in $\underline{\mathbb{N}}_2$. Since $X \notin \text{TAUT}$, we have that $\underline{\mathbb{N}}_2 \not\models (\exists x \leq |X|)(|X| = 4(x+1) \wedge Z'(x))$. Therefore, $\forall \mathit{Sound}_g$ is not true in the standard model, which is a contradiction.

($\Leftarrow$) Suppose that $\forall X(\exists Y(g(Y) = X) \Rightarrow X \in \text{TAUT})$. We want to show that $\forall \mathit{Sound}_g$ is true in $\underline{\mathbb{N}}_2$. For the sake of contradiction, we assume that $(\exists X', W, Z, Z', V', Y')(\mathit{Eval}(X', W, Z, Z') \wedge \mathit{Comp}_g(V', Y', X'))$ is true in $\underline{\mathbb{N}}_2$ and $(\exists x \leq |X'|)(|X'| = 4(x+1) \wedge Z'(x))$ is not. Since $\mathit{Comp}_g(V', Y', X')$ is true in the standard model, we conclude that $X' \in \text{TAUT}$, which is a contradiction to our original assumption. $\Box$

5.2 The Main Theorem

Remember that the uniform reduct of a proof system f is defined to be the set

$$\mathrm{U}_f = \{\varphi(\vec{x}, \vec{X}) \in \Sigma_0^B : \{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\} \text{ has polysize } f\text{-proofs}\}$$

Let f^+ be the system f augmented to allow substitution Frege rules to be applied to tautologies after exhibiting their f-proofs.

Theorem 5.21 $\mathrm{U}_{f^+} = \mathrm{TRUE}_{\Sigma_0^B}$ *iff f^+ simulates every proof system.*

Proof: ($\Leftarrow$) Suppose that f^+ simulates every proof system. Let $\varphi(\vec{x}, \vec{X}) \in \mathrm{TRUE}_{\Sigma_0^B}$. We want to show that there exists a proof system g such that $\{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\}$ have polysize g-proofs. We assume an efficient encoding of tautologies and proofs over $\{0,1\}^*$, which can be different from the one we described previously. Let p be any proof system. We modify p in order to obtain g in the following way. For every $\pi \in \{0,1\}^*, \psi \in \{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\}$, $g(0\pi) = p(\pi)$ and $g(1Y_\psi) = Y_\psi$, where Y_ψ is the encoding of ψ, and for every other string $\pi' \in \{0,1\}^*$, $g(\pi') = \top$. Clearly, g is a proof system and $\{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\}$ have polysize g-proofs. Since f^+ simulates g, we conclude that $\{\varphi(\underline{\vec{m}}, \vec{X})[\vec{n}] : \vec{m}, \vec{n} \in \mathbb{N}\}$ has polysize f^+-proofs.

($\Rightarrow$) Suppose that $\mathrm{U}_{f^+} = \mathrm{TRUE}_{\Sigma_0^B}$ holds. We show that for every proof system g, there exists a polynomial p such that

$$\forall X, Y (g(Y) = X \Rightarrow \exists Y'(f^+(Y') = X \wedge |Y'| \leq p(|Y|))).$$

Let g be any proof system. Then g satisfies (32). Therefore, $\forall Sound_g$ is true in $\underline{\mathbb{N}}_2$, by Theorem 5.20. Therefore, $\{Sound_g(X, W, Z, Z', V, Y)[\vec{n}] : \vec{n} \in \mathbb{N}\}$ has polysize f^+-proofs, by assumption.

Now, let A and B be any binary string such that $g(B) = A$ (i.e. A encodes a tautology and B is a g-proof of A), and let C be any binary string such that C encodes a computation of a Turing machine and C shows that $g(B) = A$. Furthermore, let D be a binary string such that D encodes the weight of the tautology encoded by A. Additionally, let $n_A = |A|, n_B = |B|, n_C = |C|, n_D = |D|, n_Z = n_A$ and $n_{Z'} = n_A$.

Note that the propositional formula encoded by A is in Polish notation. Thus, a propositional variable has its index written in unary notation. We write p_i for $p11\ldots1$, where $p11\ldots1$ has i many 1's. Now, we denote by $\varphi(p_0, \ldots p_{l-1})$ the formula encoded by A, where $p_0, \ldots, p_{l-1}$ are all the propositional variables in φ.

We show that there exists a binary string B' such that $f^+(B') = A$ and $|B'| \leq p(|B|)$, for some polynomial p.

The formula $Sound_g(X, W, Z, Z', V, Y)[n_A, n_D, n_Z, n_{Z'}, n_C, n_B]$ has atoms $p_0^X, \ldots, p_{n_A-2}^X, p_0^W, \ldots, p_{n_D-2}^W, p_0^Z, \ldots, p_{n_Z-2}^Z, p_0^{Z'}, \ldots, p_{n_{Z'}-2}^{Z'}, p_0^V, \ldots, p_{n_C-2}^V$ and $p_0^Y, \ldots, p_{n_B-2}^Y$, and more importantly, it has polysize f^+-proof, since we have that $Sound_g(X, W, Z, Z', V, Y) \in \mathrm{TRUE}_{\Sigma_0^B}$.

Let σ_1 be a substitution such that:

$$\begin{aligned}
&\sigma_1(p_0^X) = A(0), \ldots, \sigma(p_{n_A-2}^X) = A(n_A - 2),\\
&\sigma_1(p_0^Y) = B(0), \ldots, \sigma(p_{n_B-2}^Y) = B(n_B - 2),\\
&\sigma_1(p_0^V) = C(0), \ldots, \sigma(p_{n_C-2}^V) = C(n_C - 2),\\
&\sigma_1(p_0^W) = D(0), \ldots, \sigma(p_{n_D-2}^V) = D(n_C - 2)
\end{aligned}$$

and for every other atom p in $Sound_g(X,W,Z,Z',V,Y)[n_A,n_D,n_Z,n_{Z'},n_C,n_B]$, $\sigma_1(p)=p$. Then, we have

$$\text{(34)} \qquad \frac{Sound_g(X,W,Z,Z',V,Y)[n_A,n_D,n_Z,n_{Z'},n_C,n_B]}{Sound_g(X,W,Z,Z',V,Y)[n_A,n_D,n_Z,n_{Z'},n_C,n_B]\sigma_1}$$

by the application of the substitution rule (cf. Definition 3.43). We denote the formula $Sound_g(X,W,Z,Z',V,Y)[n_A,n_D,n_Z,n_{Z'},n_C,n_B]\sigma_1$ by $Sound_g^1$. $Sound_g^1$ is of the form

$$\text{(35)} \qquad \begin{pmatrix} Eval(X,W,Z,Z')[n_A,n_D,n_Z,n_{Z'}]\sigma_1 \\ \wedge \\ Comp_g(V,Y,X)[n_c,n_A,n_B]\sigma_1 \end{pmatrix} \to \varphi_{ta}(X,Z')[n_A,n_{Z'}]\sigma_1$$

where $\varphi_{ta}(X,Z')[n_A,n_{Z'}]\sigma_1$ is

$$\text{(36)} \qquad \bigvee_{i\leq n_A} (\underline{n}_A = 4(\underline{i}+1))[n_A] \wedge Z'(\underline{i})[n_{Z'}]$$

where we have that

$$\text{(37)} \qquad Z'(\underline{i})[n_{Z'}] =_{df} \begin{cases} p_i^{Z'} & \text{if } i \leq n_{Z'}-2 \\ \top & \text{if } i = n_{Z'}-1 \\ \bot & \text{otherwise} \end{cases}$$

Now (36) is equivalent to $Z'(\underline{j})[n_{Z'}]$, for some $j \leq n_A$. Clearly, this equivalence has polysize Frege-proof (all that is needed to be done is some pruning). Let $Sound_g^2$ denote the formula

$$\text{(38)} \qquad \begin{pmatrix} Eval(X,W,Z,Z')[n_A,n_D,n_Z,n_{Z'}]\sigma_1 \\ \wedge \\ Comp_g(V,Y,X)[n_c,n_A,n_B]\sigma_1 \end{pmatrix} \to Z'(\underline{j})[n_{Z'}]$$

Let m be the runtime required by the Turing machine M on B, which computes g and whose computation is encoded by C. Since $Comp_g(V,Y,X)[n_c,n_A,n_B]\sigma_1$, in $Sound_g^2$, is a closed tautology, we conclude that it has polysize Frege-proof, by Corollary 3.41. Thus, we obtain

$$Sound_g^3 = Eval(X,W,Z,Z')[n_A,n_D,n_Z,n_{Z'}]\sigma_1 \to Z'(\underline{j})[n_{Z'}]$$

Let σ_2 be a substitution such that $\sigma_2(p_0^Z)=p_0,\ldots,\sigma_2(p_{l-1}^Z)=p_{l-1},\sigma_2(p_l^Z)=\bot,\ldots,\sigma_2(p_{n_Z-2}^Z)=\bot$ and for every other atom p in $Sound_g^3$, $\sigma_2(p)=p$. Then, we have that

$$\text{(39)} \qquad \frac{Sound_g^3}{Sound_g^3\sigma_2}$$

by the application of the substitution rule.

Now, for each subformula φ' of φ, we substitute φ' for $p_i^{Z'}$ in $Sound_g^3\sigma_2$, where $Z'(i)$ codes the truth assignment to φ', and we set the remaining $p_j^{Z'}$ to $\bot$. Let σ_3 denote that substitution. The resulting formula has the form

$$\text{(40)} \qquad Eval^4 \to \varphi$$

We now argue that $Eval^4 = Assign^4 \wedge \varphi^4_{C_1} \wedge \ldots \varphi^4_{C_4}$ has short Frege-proof.

Remember that $Eval(X,W,Z,Z') =_{df} Assign(X,W,Z) \wedge \varphi_{C_1} \wedge \ldots \wedge \varphi_{C_4}$, where $Assign(X,W,Z) =_{df} Fla(X,W) \wedge (|Z| \leq |X|)$. Since $Assign^4$ is a closed tautology, by corollary 3.41, it has polysize Frege-proof. Now, every variable in $\varphi^4_{C_i}$ occurs in a subformula of the form

1. $\varphi^{Z'}_j \leftrightarrow \top$,
2. $\varphi^{Z'}_j \leftrightarrow \bot$,
3. $\varphi^{Z'}_j \leftrightarrow \neg\varphi^{Z'}_{j-1}$ or
4. $\varphi^{Z'}_j \leftrightarrow \varphi^{Z'}_{j-1} \star \varphi^{Z'}_l$,

which will turn into tautologies, by σ_3, with short Frege-proofs. As an example, let us look at $\varphi^4_{C_2}$. $\varphi_{c_2}\sigma_1\sigma_2$ is of the form

$$\bigwedge_{j,k \leq n_A} Subf(X,\underline{j},\underline{k},W)[n_A,n_D]\sigma_1\sigma_2 \wedge XEnc_{\neg}(X,\underline{j})[n_A]\sigma_1\sigma_2 \rightarrow \psi_1$$

where

$$\psi_1 = \begin{cases} \bot \leftrightarrow \bot & \text{if } j = n_{Z'} \\ \top \leftrightarrow \neg p^{Z'}_{j-1} & \text{if } j = n_{Z'} - 1 \\ p^{Z'}_j \leftrightarrow \neg p^{Z'}_{j-1} & \text{if } j \leq n_{Z'} - 2 \end{cases}$$

$\varphi^4_{C_2}$ can be shown to be equivalent to

$$\bigwedge_{j,k \leq n_A - 2} \underbrace{Subf(X,\underline{j},\underline{k},W)[n_A,n_D]\sigma_1\sigma_2 \wedge XEnc_{\neg}(X,\underline{j})[n_A]\sigma_1\sigma_2 \rightarrow \psi_2}_{\psi} \tag{41}$$

by a short Frege-proof, where ψ_2 is the tautology $\neg\varphi' \leftrightarrow \neg\varphi'$, which also has short Frege-proof and where φ' is the subformula of φ corresponding to $Subf(X, j-1, k)$. Since the tautology ψ_2 has short Frege-proof, we conclude that ψ has short Frege-proof. Then, (41) has short Frege-proof. Similarly, for $\varphi^4_{C_1}, \varphi^4_{C_3}$ and $\varphi^4_{C_4}$. Thus, $Eval^4$ has short Frege-proof.

We conclude that φ has an f^+-proof polynomial in the length of B. $\square$

6 Conclusion

In this dissertation, we carried out a detailed proof of the equivalence between the existence of an optimal proof system and the existence of a proof system whose uniform reduct is the set of all true Σ^B_0-formulae. In this regard, we described how to encode Polish propositional formulae into binary strings (or sets, more precisely) and Σ^B_0-defined Polish propositional formulae. Our description of how to encode truth assignments to Polish propositional formulae is partly from Cook and Nguyen's book, "Logical Foundations of Proof Complexity". Combining the ideas of Papadimitriou's [Pap94] and Cook and Nguyen's, of how to encode a Turing machine computation, we described how to encode a polytime Turing machine computation on a given input and provided a Σ_0^B-formula that captures such computation. We also gave a Σ_0^B-formulation of the Reflection Principle.

In Cook's sketch of the main theorem [Coo06], he used $Z(0)$ instead of our $Z(\underline{n})$, where $\underline{n}$ is a natural number such that the length of the encoding of the Polish formula under consideration is equal to $(n+1)$, to represent the truth value of a Polish formula. This allowed him to do a direct proof without having to go through the equivalence between (36) and $Z'(\underline{n})[n_{Z'}]$. This suggests that, instead of Polish formulae, we could use reverse Polish formulae. Thus, $Z(0)$ would then represent the truth value of the reverse Polish formula under consideration.

As we have seen, if one can show that a proof system f is optimal, then separating NP from coNP boils down to showing if there exists a true Σ_0^B-formula $\varphi(\vec{X})$ such that $\{\varphi(\vec{X})[\vec{n}] : \vec{n} \in \mathbb{N}\}$ is hard for f. Thus, a possible future direction would be to investigate the uniform reducts of propositional proof systems whose no strong lower bounds are not known yet: Frege, extended Frege, etc. For example, one may look for properties of uniform reducts of those systems which might help distinguish them from $\mathrm{TRUE}_{\Sigma_0^B}$.

References

[BBP95] Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. Are there hard examples for Frege systems? In *Feasible mathematics, II (Ithaca, NY, 1992)*, volume 13 of *Progr. Comput. Sci. Appl. Logic*, pages 30–56. Birkhäuser Boston, Boston, MA, 1995.

[BdG98] Shai Ben-david and Anna Gringauze. On the existence of optimal propositional proof systems and oracle-relativized propositional logic. Technical report, Electronic Colloquium on Computational Complexity, 1998.

[Bec05] Arnold Beckmann. Uniform proof complexity. *J. Logic Comput.*, 15(4):433–446, 2005.

[BP96] Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *37th Annual Symposium on Foundations of Computer Science (Burlington, VT, 1996)*, pages 274–282. IEEE Comput. Soc. Press, Los Alamitos, CA, 1996.

[BP98] Sam Buss and Toniann Pitassi. Resolution and the weak pigeonhole principle. In *Computer science logic (Aarhus, 1997)*, volume 1414 of *Lecture Notes in Comput. Sci.*, pages 149–156. Springer, Berlin, 1998.

[BP01] Paul Beame and Toniann Pitassi. Current trends in theoretical computer science. chapter Propositional proof complexity: past, present, and future, pages 42–70. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2001.

[Bus87a] Samuel R. Buss. The boolean formula value problem is in alogtime. In *in Proceedings of the 19-th Annual ACM Symposium on Theory of Computing*, pages 123–131, 1987.

[Bus87b] Samuel R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *J. Symbolic Logic*, 52(4):916–927, 1987.

[Bus02] Samuel R. Buss. Some remarks on lengths of propositional proofs. *Archive for Mathematical Logic*, 34:377–394, 2002.

[CN10] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[Coo03] Stephen Cook. The importance of the p versus np question. *J. ACM*, 50:27–29, January 2003.

[Coo06] Stephen Cook. Comments on Beckmann's uniform reducts. *CoRR*, abs/cs/0601086, 2006.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symbolic Logic*, 44(1):36–50, 1979.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[Hak85] A. Haken. The Intractability of Resolution. *Theoretical of Computer Science*, 39:297–308, 1985.

[KP89a] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *J. Symbolic Logic*, 54(3):1063–1079, 1989.

[KP89b] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *J. Symbolic Logic*, 54(3):1063–1079, 1989.

[MT98] Jochen Meßner and Jacobo Torán. Optimal proof systems for propositional logic and complete sets. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '98, pages 477–487, London, UK, 1998. Springer-Verlag.

[Pap94] Christos M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[Raz04] Ran Raz. Resolution lower bounds for the weak pigeonhole principle. *J. ACM*, 51(2):115–138 (electronic), 2004.