On meta complexity of propositional formulas and propositional proofs

Pavel Naumov
Department of Mathematics
and Computer Science
McDaniel College
Westminster, MD 21157

pnaumov@mcdaniel.edu

March 4, 2008

Abstract

A new approach to defining complexity of propositional formulas and proofs is suggested. Instead of measuring the size of these syntactical structures in the propositional language, the article suggests to define the complexity by the size of external descriptions of such constructions. The main result is a lower bound on proof complexity with respect to this new definition of complexity.

1 Introduction

Propositional proof complexity is concerned with asymptotic analysis of the minimal proof size as a function of the tautology size. The details of this analysis depend on the three major factors: the logical formalism that is used, the definition of the proof size, and the definition of the formula size. A variety of formalisms has been previously considered; among them are Frege and extended Frege [1], limited-depth Frege [9, 7], resolution [3], and cutting plane systems [2]. Two major ways to measure the size of a propositional proof is to count the number of symbols and the number of steps in the proof. The most popular way to measure a formula size is to count the number of symbols in this formula. Clearly, alternative definitions of the formula size are possible. For example, formula size could be defined through minimal size of binary decision diagram ("BDD") representing the formula. Nevertheless, such definitions are hardly used.

1.1 Tautology descriptions

The mentioned above measurements of tautologies and proofs sizes are based on the internal descriptions of such objects in the propositional logic. These descriptions, in turn, depend on the choice of syntactical primitives in the propositional language. If these primitives are changed, the size of the formula could change as well. For example, consider propositional pigeonhole principle:

$$\bigwedge_{i\leq n}\bigvee_{j<n} p_{i,j} \rightarrow \bigvee_{i_1\neq i_2}\bigvee_{j}(p_{i_1,j}\wedge p_{i_2,j}). \tag{1}$$

Traditionally, multi-argument symbols $\bigwedge$ and $\bigvee$ are viewed as meta abbreviations for multiple applications of binary conjunctions and disjunctions. In this case, the size of the formula (1) is $O(n^3)$. Yet, if we modify propositional language to include symbols $\bigwedge$ and $\bigvee$, then the size of formula (1) becomes $O(\log n)$ – the number of digits required to write number n.

It is worth pointing out that although the tradition to use binary logical connectives has a long history, its main justification is probably the Boole's desire to present propositional logic in the algebraic form. Putting algebraic connection aside, there is no internal logical reason to restrict ourselves by binary and unary syntactical primitives. In fact it is quite common in practice to introduce non-standard notations to describe complex boolean expression just like we did in formula (1). Use of BDDs to describe complex propositional formulas in computer-based model checking is another example of turning to non-standard logical notations to shorten a representation of a boolean expression.

To make the notion of the tautology size less dependent on the particular choice of syntactical primitives we can instead use the notion of tautology complexity which would be defined as the shortest description of the tautology in a certain class of notations. The most "robust" of these classes of notations is, probably, the approach under which tautology is denoted by a minimal Turing Machine that can generate this tautology. This, essentially, is Kolmogorov complexity applied to the domain of propositional formulas. It is a valid approach to formula complexity, but it does not make sense in the proof complexity, where we study dependency of the proof size on the tautology size. Indeed, by incorporating a side computation in the Turing machine describing tautology, we can easily achieve the result when it would be very hard to prove the tautology generated by the Turing machine. For example, for any recursive function $f(n)$ we can define Turing Machine TM_f that on any input n looks at all propositional proofs of size no more than $f(n)$ and finds minimal m such that tautology $\top^m \equiv \top\wedge\cdots\wedge\top$ is not proven by any of these proofs. The Turing machine TM_f then returns tautology $\top^m$ as output. Kolmogorov complexity of such tautology is determined by the size of description of n which is in $O(\log n)$, yet the minimal size of the proof of this tautology is at least $f(n)$.

Another approach to defining the class of compact notations for propositional formulas is suggested by Krajíček in his work on implicit proofs [5, 4, 6]. Essentially, he restricts Turing machines to digital circuits in the above definition. To be more specific, the digital circuit outputs consecutive fragments of

binary formula description on the standard enumeration of all possible binary inputs. The complexity of a boolean formula is defined to be the minimal size of a digital circuit that generates it. This is an interesting approach from the computer science prospective and because of its obvious connection to digital circuit complexity.

In this article we develop another class of notations for boolean formulas that we call *boolean recursions*. These notations meant to generalize multi-argument conjunction and disjunction mentioned above. Namely, we notice that these multi-argument connectives could be defined recursively. For example, propositional formula $\bigvee_{i=1}^{n} p_i$ could be defined by recursion over parameter n as follows:

$$\bigvee_{i=1}^{n+1} p_i = (\bigvee_{i=1}^{n} p_i) \vee p_{n+1}, \qquad \bigvee_{i=1}^{0} p_i = \bot.$$

Such recursion together with the specific value of n could be use to described formula $\bigvee_{i=1}^{n} p_i$. The size of this description is $O(\log n)$ because recursion description has a constant size and only $O(\log n)$ symbols are needed to specify the value of the parameter n, if standard binary notations for integers are used. In Section 2.3, we will give a more detailed definition of boolean recursions that we will use to describe propositional formulas. We will keep our presentation general enough to allow representations of integers possibly different from standard binary notations.

1.2 Meta proof

One can view boolean recursions as a new and very powerful primitive syntactical construction that we add to the propositional language. Yet, another approach is to view boolean recursions as "meta" descriptions of traditional propositional formulas. We use the word "meta" to stress the fact that the description is given outside of the standard propositional language. If our goal is to compare tautology complexity to its proof complexity, then one would expect a similar measure of proof complexity through the size of its meta description to be introduced.

What could be viewed as a proper "meta" description of the proof? Our informal answer to this question is: whatever a mathematician will write on the blackboard when faced with a task of presenting a propositional proof. Let us see how this informal answer can be translated into a formal one. We will start with a discussion of two extreme alternatives.

According to one of them, a meta description of a proof is simply a sequence of the meta descriptions of the formulas, where the formulas form a proof in the traditional sense. This approach "compresses" each formula in the proof, but leaves the number of proof steps unchanged. Although reasonable, this approach leads to a very weak notion of meta proof. For example, if a propositional proof is using formulas $\phi_1, \ldots, \phi_n$ that could be obtained one from another by rearranging arguments of conjunctions, one would want the meta proof to be able

to make this observation and not to re-prove each of these formulas separately. Our current meta proofs will not be able to do that.

The other extreme is based on Kolmogorov complexity. We could simply say that proof described by a Turing machine that outputs the proof. This approach has two major flaws. First, mathematicians hardly would accept a proof-generating algorithm as a "meta proof" without a justification that this algorithm terminates and produces a valid proof. Second, *exhaustive search* algorithm satisfies this definition of the meta proof, thus trivializing the notion of proof complexity. Krajíček's notion of *implicit proof* is an attempt to fix these flaws by replacing Turing machines with digital circuits and requiring to include into the implicit proof a propositional proof of correctness for such circuit (see [5]).

In this work we want to propose a formal notion of *meta proof* that could be viewed as an alternative to implicit proofs. Namely, consider any meta theory capable of formalizing and reasoning about the syntax of propositional logic. To keep the presentation simple, we will assume that meta theory is Peano Arithmetic (PA). Let $Taut(x)$ be a Gödel-style provability predicate for propositional logic. Informally, $Taut(\ulcorner\phi\urcorner)$ is an arithmetical formula that states that "proposition ϕ is provable in the propositional logic". By meta proof of proposition ϕ we will mean a proof of $Taut(\ulcorner\phi\urcorner)$ in PA. One might criticize this definition of meta proofs for not being constructive. Indeed, we only establish existence of the proof without giving any instructions on how to construct such a proof. Our objection to this critique is that once the existence of the proof is established, the proof itself could be found through an exhaustive search algorithm, which now would be guaranteed to terminate.

1.3 Proof complexity

Size of an arithmetical proof is the total number of symbols in the proof. By meta complexity of a propositional proof we will mean the minimal size of an arithmetical proof of existence of such a propositional proof. In the other words, proof complexity of proposition ϕ is just the minimal size of the proof of $Taut(\ulcorner\phi\urcorner)$ in PA.

The arithmetical predicate $Taut(x)$ could be written for different propositional systems, such as Frege, extended Frege, resolution, etc. Clearly, all such predicates are provably equivalent in PA. Thus, the introduced notion of meta proof complexity is system-independent modulo $O(1)$. This contrasts with the traditional notion of proof complexity that varies from one propositional system to another. Of course, meta proof complexity might, in turn, depend on the first-order formalization of PA, but one can easily see that the main results of this paper do not depend on the particular choice of formalism used for PA.

The main result of this article, Theorem 1, is a lower bound on meta proof complexity. In case when standard binary notations are used for integers this lower bound is super polynomial (see Corollary 1). That is, for any polynomial $p(n)$ there is a propositional tautology ϕ that could be defined through a boolean recursion of size n such that $Taut(\ulcorner\phi\urcorner)$ has no PA proofs of size no larger than

$p(n)$.

It should be pointed out that this new, more syntax-independent, approach to measuring formula and proof complexity is induced by the interest to pure proof complexity, not its applications to computational complexity. Application of the presented here results to computational complexity appears to be a hard task especially because meta complexity approach modifies not only the way complexity of proofs is measured but it also modifies the way formula size is defined. At the same time, by measuring propositional proof complexity by the size of PA proofs, meta proof complexity builds a bridge that connects propositional proof complexity with the first order proof complexity [8].

1.4 Proof Idea

To prove Theorem 1, we need to construct a propositional tautology that has no short proofs. We construct this tautology by finding a way to write a propositional formula that essentially says *I have no short proofs*. It is easy to see that such formula can not be false, but, if true, it can not have short proofs.

2 Formal Setting of the Problem

2.1 Binary Representation of Formulas

We assume that the alphabet of the propositional language consists of parentheses, boolean connectives, propositional variables $p_0, p_1, \ldots$, and comma sign. Let τ be an arbitrary mapping of the symbols in this alphabet into binary strings. We assume that this mapping is naturally extended to a function from words in this alphabet to binary strings. The exact choice of translation τ is not important, but we assume that it is fixed throughout this article and satisfies the following properties:

1. The translation of each symbol starts with 1.
2. The translation is an injective function from words into binary strings.
3. For an arbitrary propositional variable p_n, translation $\tau(p_n)$ is $\alpha\beta\gamma$, where binary strings α and γ do not depend on the value of parameter n and β is a representation of integer n in binary (base two) numerical system.

Definition 1 *Size $|\phi|$ of a propositional formula ϕ is the length of string $\tau(\phi)$. Size $|\pi|$ of a propositional proof π is the length of string $\tau(\pi)$.*

Lemma 1

$$|p_n| \in O(\log n).$$

Proof. See condition 3 above. □

We also will consider a mapping θ of the symbols in the first-order language of Peano Arithmetic into binary strings. This mapping, just like τ, could be

extended to a mapping from words in the arithmetical language into binary strings. Throughout this article mapping θ is also assumed to be fixed and satisfying conditions 1 and 2 above. Although condition 3 could be modified from one on propositional variables into a one on first-order variables, the results presented in this paper do not require translation θ to satisfy condition 3.

Definition 2 *Size $|\phi|$ of a first order formula ϕ is the length of string $\theta(\phi)$. Size $|\pi|$ of a arithmetical proof π is the length of string $\theta(\pi)$.*

2.2 Meta Proof Complexity

By Gödel numbers people usually mean a way to assign a unique integer number to each arithmetical formula. Ordinarily, the exact way this assignment is done is not important as long as there is a set of primitive-recursive functions on integer numbers that correspond to syntactical operations on formulas. Here we are going to deal with sizes of arithmetical formulas, so we would like to adopt a system of Gödel numbers in which sizes of these numbers could be easily determined based on the size of the formulas. For this purpose, we assume that Gödel number $\ulcorner\phi\urcorner$ of any arithmetical formula ϕ is the integer whose binary representation is $\theta(\phi)$. Note that $\theta(\phi)$ could be interpreted as a number in binary notations because, by condition 1 above, binary string $\tau(\phi)$ always starts with 1. It is important to observe that Gödel number of an arithmetical formula ϕ uses exactly $|\phi|$ binary digits. Hence, the size of an arithmetical formula with Gödel number x is $\lfloor \log x \rfloor + 1$. Therefore,

Lemma 2 *Size of an arithmetical formula ϕ is $\lfloor \log(\ulcorner\phi\urcorner) \rfloor + 1$.* □

Bear in mind, however, that standard formalization of Peano Arithmetic does not use binary numerals. Instead, it provides two atomic numerals: 0 and 1, and the other numerals should be specified as terms. If $d_1 d_2 d_3 \dots d_n$ is an integer number in binary notations, then it can be written as an arithmetical term

$$((d_1 \times (1+1) + d_2) \times (1+1) + d_3) \times (1+1) + \dots + d_n.$$

The size of this term is $O(n)$. Assuming such encoding of binary numbers, we could assert

Lemma 3 *For any given arithmetical formula $P(x)$,*

$$|P(\ulcorner\phi\urcorner)| \in O(|\phi|).$$

Proof. By Lemma 2, $|P(\ulcorner\phi\urcorner)| \in O(\log(\ulcorner\phi\urcorner)) = O(|\phi|)$. □

Let $Pr(x)$ be the Gödel provability predicate for Peano Arithmetic built using the described above Gödel numbers. That is, $Pr(\ulcorner\phi\urcorner)$ is an arithmetical statement that claims that arithmetical formula ϕ is provable in Peano Arithmetic.

Lemma 4 *There is a monotonic polynomial $p_1(n)$ such that $|Pr(\ulcorner\alpha\urcorner)| \le p_1(|\alpha|)$ for any arithmetical formula α.*

Proof. By Lemma 3, polynomial $p_1(n)$ can be chosen to be a linear function. $\square$

We will also consider a variation of the provability predicate, bounded provability predicate, $Pr_n(\ulcorner\phi\urcorner)$ that claims that arithmetical formula ϕ has a proof π in Peano Arithmetic such that $|\pi| \leq n$. In addition, we will introduce Gödel-like numbering of *propositional* formulas in such a way that for any propositional formula ϕ, its Gödel number $\ulcorner\phi\urcorner$ is the integer whose binary (in the numerical system with base two) representation is $\tau(\phi)$. Propositional provability predicate $Taut(\ulcorner\phi\urcorner)$ is an arithmetical statement that claims that propositional formula ϕ is provable in propositional logic. Finally, we will use notation $PA \vdash_n \phi$ to say that arithmetical formula ϕ has a proof π in Peano Arithmetic such that $|\pi| \leq n$.

Definition 3 *Meta proof complexity $||\phi||_{PA}$ of a propositional formula ϕ is the minimal size of a proof of the statement $Taut(\ulcorner\phi\urcorner)$ in Peano Arithmetic.*

2.3 Boolean Recursions

Let α and $\beta(q,r)$ be propositional formulas (boolean expressions), and q and r be two different propositional variables that occur in β. A single one-parameter boolean recursion is a formalism that we suggest for specifying an infinite sequence of boolean expressions $b[0], b[1], b[2], \ldots$ that could be informally defined as $\alpha, \beta(\alpha, p_0), \beta(\beta(\alpha, p_0), p_1), \ldots$, where $p_0, p_1, p_2, \ldots$ is the list of all propositional variables. More formally, it can be specified as

$$\left\{ \begin{array}{lcl} b[n+1] & = & \beta(b[n]/q, p_n/r), \\ b[0] & = & \alpha. \end{array} \right. \tag{2}$$

An example of such sequence is $b[n] = \bigwedge_{i=0}^{i<n} p_i$, it can be defined by the following boolean recursion:

$$\left\{ \begin{array}{lcl} b[n+1] & = & b[n] \wedge p_n, \\ b[0] & = & \top. \end{array} \right.$$

In this example, $\alpha = \top$ and $\beta(q,r) = q \wedge r$.

In a more general case, we will allow simultaneous recursion over two parameters:

$$\left\{ \begin{array}{ll} b[n+1, m+1] & = \delta(b[n+1, m]/q_1, b[n, m+1]/q_2, b[n, m]/q_3, p_n/r_1, p_m/r_2), \\ b[n+1, 0] & = \gamma(b[n, 0]/q, p_n/r), \\ b[0, m+1] & = \beta(b[0, m]/q, p_m/r), \\ b[0, 0] & = \alpha. \end{array} \right.$$

Technically, two-parameter boolean recursion is formally defined by specifying four boolean expressions: α, $\beta(q,r)$, $\gamma(q,r)$, and $\delta(q_1, q_2, q_3, r_1, r_2)$ that correspond to four different recursion cases. Generally speaking, we will allow boolean recursions with arbitrary number of parameters $b[n_1, \ldots, n_l]$.

We will also allow systems of boolean recursions. For example, a system of two single-parameter recursions has the form

$$\left\{\begin{array}{l} \left\{\begin{array}{lcl} a[n+1] & = & \beta(a[n]/q_1, b[n]/q_2, p_n/r), \\ a[0] & = & \alpha, \end{array}\right. \\ \\ \left\{\begin{array}{lcl} b[n+1] & = & \delta(a[n]/q_1, b[n]/q_2, p_n/r), \\ b[0] & = & \gamma. \end{array}\right. \end{array}\right.$$

In this paper, by a system of boolean recursions $\mathcal{R}$ we mean a finite system of boolean recursions such that each recursion in the system has a finite number of arguments. Informally, the size $|\mathcal{R}|$ of system $\mathcal{R}$ is the number of symbols in the (finite) description of the system. Formally, system $\mathcal{R}$ is a finite sequence of equalities, by the size $|\mathcal{R}|$ of system $\mathcal{R}$ we mean the total number of symbols in all these equalities. Let $b[n_1, n_2, \ldots, n_l]$ be one of the recursions specified by system $\mathcal{R}$. Any particular boolean expression in this recursion can be described by system $\mathcal{R}$, name of recursion b and values of integer parameters $n_1, n_2, \ldots, n_l$. By $|b[n_1, n_2, \ldots, n_l]|_{\mathcal{R}}$ we mean the size of such a description.

We will assume that "the standard" way to describe an integer is using binary (based two) notations. However, to state our main result in a more general form, we will allow for non-standard descriptions. For example, one can describe integer by an arithmetical expression, by an expression that includes exponential functions, etc. Let $decode : \{0,1\}^* \to \mathbb{Z}$ be an arbitrary injective function from binary strings of arbitrary length into integers. We define

$$e(l) = max\{decode(s) : length(s) = l\}.$$

It will be assumed throughout the rest of the paper that function $e(l)$ could be defined in Peano Arithmetic. That is, there is an arithmetical formula $\phi(x, y)$ such that $e(x) = y$ if and only if $\phi(x, y)$. Also, we assume that $E(l)$ is a binary string such that $length(decode(E(l))) = e(l)$. By $|n|$ we will mean the length of the shortest string s such that $decode(s) = n$. Note that for standard encoding of integer numbers (in binary notations), we have $e(l) = 2^l - 1$.

Lemma 5 *For any system of boolean recursions $\mathcal{R}$ and any boolean expression $b[n_1, n_2, \ldots, n_m]$ defined by system $\mathcal{R}$,*

$$|b[n_1, n_2, \ldots, n_m]|_{\mathcal{R}} = O(|\mathcal{R}| + |n_1| + \cdots + |n_m|).$$

Finally, if formula ϕ is a boolean combination

$$\alpha(b^1[n_1^1, n_2^1, \ldots, n_l^1], b[n_1^2, n_2^2, \ldots, n_l^2], \ldots, b[n_1^m, n_2^m, \ldots, n_l^m])$$

of formulas $b^1[n_1^1, n_2^1, \ldots, n_l^1], b[n_1^2, n_2^2, \ldots, n_l^2], \ldots, b[n_1^m, n_2^m, \ldots, n_l^m]$ defined by a system of boolean recursions $\mathcal{R}$, then by $|\phi|_{\mathcal{R}}$ we will mean

$$|\alpha| + \sum_{i=1}^{m} |b[n_1^i, n_2^i, \ldots, n_l^i]|_{\mathcal{R}}.$$

2.4 The Main Result

In this section we state an upper lower bounds on meta proof complexity.

Definition 4 *For any two functions $f, g : \mathbb{N} \to \mathbb{N}$, we say that function g polynomially dominates function f if for any two polynomials $p(x)$ and $q(x)$ there is $N \in \mathbb{N}$ such that for any $n > N$,*

$$p(f(q(n))) < g(n).$$

We will use notation $f \prec g$ to state that function g polynomially dominates function f. Here are some trivial examples of polynomial domination: $x \prec 2^x$, $2^x \prec 2^{2^x}$, $2^{2^x} \prec 2^{2^{2^x}}$, etc.

Recall that above we have defined function e.

Theorem 1 *For any function f such that $f \prec e$ there is a propositional tautology $b[n_1, n_2, \ldots, n_m]$ defined by a system of boolean recursions $\mathcal{R}$ such that*

$$||b[n_1, n_2, \ldots, n_m]||_{PA} > f(|b[n_1, n_2, \ldots, n_m]|_{\mathcal{R}}).$$

Corollary 1 *If standard binary notations are used to represent integers, then for any polynomial p there is a propositional tautology $b[n_1, n_2, \ldots, n_m]$ defined by a system of boolean recursions $\mathcal{R}$ such that*

$$||b[n_1, n_2, \ldots, n_m]||_{PA} > p(|b[n_1, n_2, \ldots, n_m]|_{\mathcal{R}}).$$

The rest of the article is dedicated to the proof of Theorem 1.

3 Boolean Encoding of Arithmetical Proofs

Let $p_0, p_1, p_2, \ldots$ be variables in the propositional language. Every valuation $*$ can be uniquely identified with an infinite binary sequence $p_0^*, p_1^*, p_2^*, \ldots$. Our goal for this section is to write a system of boolean recursions $\mathcal{R}$ that defines a propositional formula Δ_n^α that says that $p_0^*, \ldots, p_{e(n)-1}^*$ is not a translation of an arithmetical proof of statement α. The existence of such a propositional formula is not surprising. We just should be careful enough to garantee that $|\Delta_n^\alpha|_{\mathcal{R}}$ is sufficiently small. We start with several primitives that later will be used to define system $\mathcal{R}$ and formula Δ_n^α.

3.1 Index Arithmetic

Definition 5 *Let the propositional formula $eq[i, j]$ be defined by the following boolean recursion:*

$$\begin{cases} eq[0,0] & = & \top, \\ eq[0, j+1] & = & \bot, \\ eq[i+1, 0] & = & \bot, \\ eq[i+1, j+1] & = & eq[i, j]. \end{cases} \tag{3}$$

Thus, for any non-negative integers i and j, formula the $eq[i,j]$ is a variable-free propositional formula. Depending on the values of i and j it is equal to either $\bot$ or $\top$.

Lemma 6 *For any non-negative integers i and j, the propositional formula $eq[i,j]$ is true if and only if $i = j$.* □

Definition 6 *Let the propositional formula $less[i,j]$ be defined by the following boolean recursion:*

$$\begin{cases} less[0,0] & = & \bot, \\ less[0,j+1] & = & \top, \\ less[i+1,0] & = & \bot, \\ less[i+1,j+1] & = & less[i,j]. \end{cases} \tag{4}$$

Lemma 7 *For any non-negative integers i and j, the propositional formula $less[i,j]$ is true if and only if $i < j$.* □

Definition 7 *Let the propositional formula $addeq[i,j,k]$ be defined by the following boolean recursion:*

$$\begin{cases} addeq[0,0,0] & = & \top, \\ addeq[0,0,k+1] & = & \bot, \\ addeq[0,j+1,0] & = & \bot, \\ addeq[0,j+1,k+1] & = & eq[j,k], \\ addeq[i+1,0,0] & = & \bot, \\ addeq[i+1,0,k+1] & = & eq[i,k], \\ addeq[i+1,j+1,0] & = & \bot, \\ addeq[i+1,j+1,k+1] & = & addeq[i+1,j,k]. \end{cases} \tag{5}$$

Lemma 8 *For any non-negative integers i, j, and k, the propositional formula $addeq[i,j,k]$ is true if and only if $i + j = k$.* □

Definition 8 *Let the propositional formula $prev[i,j]$ be defined by the following boolean recursion:*

$$\begin{cases} prev[0,0] & = & \bot, \\ prev[0,j+1] & = & eq[0,j], \\ prev[i+1,0] & = & \bot, \\ prev[i+1,j+1] & = & prev[i,j]. \end{cases} \tag{6}$$

Lemma 9 *For any non-negative integers i and j, the propositional formula $prev[i,j]$ is true if and only if $i + 1 = j$.* □

3.2 Bounded Quantifier

Definition 9 *For an arbitrary propositional formula $\phi(i)$, with an integer parameter i, and an arbitrary non-negative integer n we define propositional formula $\bigvee_{i=0}^{n} \phi(i)$, with parameter n, by a boolean recursion as following:*

$$\begin{cases} \bigvee_{i=0}^{0} \phi(i) & = & \bot, \\ \bigvee_{i=0}^{n+1} \phi(i) & = & \phi(n) \vee \bigvee_{i=0}^{n} \phi(i). \end{cases} \tag{7}$$

Lemma 10 *For any propositional formula $\phi(i)$ with an integer parameter i, any non-negative integer n, and any valuation *, formula $(\bigvee_{i=0}^{n} \phi(i))^*$ is true if and only if there is an integer i such that $0 \leq i < n$ and $(\phi(i))^*$ is true.* □

Note that propositional formula $\phi(i)$ is a parameter in the expression $\bigvee_{i=0}^{n} \phi(i)$ in the sense that Definition 9 defines different formula $\bigvee_{i=0}^{n} \phi(i)$ for each propositional formula $\phi(i)$. Sometimes, we will use $\bigvee$ inside of another definition. For example, we might write

$$b[n] = \bigvee_{i=0}^{n} \alpha(b[i]).$$

Formally, this expression should be interpreted as the boolean recursion

$$\left\{ \begin{array}{lcl} b[0] & = & \bot, \\ b[n+1] & = & \alpha(b[n]) \vee b[n]. \end{array} \right.$$

We will also encounter slightly more complicated definitions involving $\bigvee$. These will have the form

$$b[n] = \beta(\bigvee_{i=0}^{n} \alpha(b[i])).$$

To write this definition as a boolean recursion, we introduce an auxiliary formula $a[n] = \bigvee_{i=0}^{n} \alpha(b[i])$, which can be defined through a boolean recursion as follows:

$$\left\{ \begin{array}{lcl} a[0] & = & \bot, \\ a[n+1] & = & \alpha(\beta(a[n])) \vee \beta(a[n]). \end{array} \right.$$

Formula $b[n]$ should be viewed now just as an abbreviated notation for $\beta(a[n])$. The use of nested instances of $\bigvee$ can be interpreted similarly by introduction of a new recursively defined boolean formula for every nested instance of $\bigvee$.

Definition 10 *For any non-negative integers i, j, i', j',*

$$subeqsub[i, j, i', j'] = \bigvee_{\Delta=0}^{j} (addeq[i, \Delta, j] \wedge addeq[i', \Delta, j']).$$

Lemma 11 *For any non-negative integers i, j, i', j', the propositional formula $subeqsub[i, j, i', j']$ is true if and only if $i \leq j$, $i' \leq j'$, and $j - i = j' - i'$.* □

3.3 Segments

Definition 11 *For any boolean string $\beta = b_0 b_2 \ldots b_{k-1}$, let boolean expression $PATTERN_\beta[i, j]$ be*

$$\bigvee_{m_1=0}^{j} \bigvee_{m_2=0}^{j} \cdots \bigvee_{m_{k-1}=0}^{j} prev[i, m_1] \wedge prev[m_1, m_2] \wedge \cdots \wedge prev[m_{k-1}, j] \wedge$$

$$\wedge (p_i \leftrightarrow b_0) \wedge (p_{m_1} \leftrightarrow b_1) \wedge \cdots \wedge (p_{m_{k-1}} \leftrightarrow b_{k-1}).$$

Note that the boolean string β is a meta-parameter of formula $PATTERN_\beta[i,j]$ in the sense that the above definition specifies a separate boolean expression for each boolean string β. The size of this expression increases as a function of the length of string β.

Lemma 12 *For any boolean string $\beta = b_0 b_1 \ldots b_{k-1}$, any non-negative integers i and j, and any valuation of propositional variables *, boolean expression $PATTERN_\beta[i,j]$ is true if and only true if $j+k=i$, $p^*_j = b_0, p^*_{j+1} = b_1, \ldots, p^*_{j+(k-1)} = b_{k-1}$.* □

Definition 12

$$segeqseg[i,j,i',j'] \;=\; subeqsub[i,j,i',j'] \wedge \neg \bigvee_{x=0}^{j} \bigvee_{x'=0}^{j'} \neg(subeqsub[i,x,i',x'] \rightarrow (p[x] \leftrightarrow p[x'])).$$

Lemma 13 *For any non-negative integers i,j,i',j', the propositional formula $segeqseg[i,j,i',j']$ is true if and only if $i \leq j$, $i' \leq j'$, and the binary string $p^*_i, p^*_{i+1}, \ldots, p^*_{j-1}$ is equal to the string $p^*_{i'}, p^*_{i'+1}, \ldots, p^*_{j'-1}$.* □

3.4 First-Order Syntax

Next, we will use the defined above primitives to recursively define boolean expresions $Variable[k,n]$, $Term[k,n]$, $Atom[k,n]$, and $Formula[k,n]$ that state that the binary string $p^*_k, \ldots, p^*_{n-1}$ is a θ-translation of an atomic predicate formula, a variable, and a predicate formula correspondingly. Below, we explicitly define a somewhat more complex expression $Formula[k,n]$, the other two expressions can be defined in a similar fashion.

Definition 13 *For any two non-negative integers n and k,*

$$\begin{aligned} Formula[k,n] \;=\; & Atom[k,n] \vee \\ & (less[k,n] \wedge \bigvee_{k_1=0}^{n} \bigvee_{k_2=0}^{n} \bigvee_{k_3=0}^{n} \bigvee_{k_4=0}^{n} (PATTERN_{\theta(\prime(\prime)} [k,k_1] \wedge \\ & \quad \wedge Formula[k_1,k_2] \wedge PATTERN_{\theta(\prime\rightarrow\prime)}[k_2,k_3] \wedge \\ & \quad \wedge Formula[k_3,k_4] \wedge PATTERN_{\theta(\prime)\prime)}[k_4,n])) \quad \vee \\ & (less[k,n] \wedge \bigvee_{k_1=0}^{n} \bigvee_{k_2=0}^{n} \bigvee_{k_3=0}^{n} \bigvee_{k_4=0}^{n} (PATTERN_{\theta(\prime\forall\prime)}[k,k_1] \wedge \\ & \quad \wedge Variable[k_1,k_2] \wedge PATTERN_{\theta(\prime(\prime)}[k_2,k_3] \wedge \\ & \quad \wedge Formula[k_3,k_4] \wedge PATTERN_{\theta(\prime)\prime)}[k_4,n])). \end{aligned}$$

Lemma 14 *For any non-negative k and n and any valuation $*$, the boolean expression $Formula[k,n]$ is true iff $k<n$ and the binary string $p^*_k, p^*_{k+1}, \ldots, p^*_{n-1}$ is a θ-translation of an arithmetical formula.* □

3.5 Encoding of Proofs

The full list of Peano Arithmetic axioms consists of propositional axiom schemata, first-order axiom schemata, and arithmetic-specific axioms and axiom schemata. For each propositional axiom or scheme we can write a boolean expression $A[k,n]$ that says that the binary string $p^*_k, p^*_{k+1}, \ldots, p^*_{n-1}$ is equal to a translation of an appropriate axiom under encoding θ. For example, the propositional axiom schemata $(\phi \to (\psi \to \phi))$ can be encoded by the following boolean expression:

$$\begin{aligned} A[k,n] \;=\; & \bigvee_{k_1=0}^{n} \bigvee_{k_2=0}^{n} \bigvee_{k_3=0}^{n} \bigvee_{k_4=0}^{n} \bigvee_{k_5=0}^{n} \bigvee_{k_6=0}^{n} (\\ & \quad PATTERN_{\theta(\prime(\prime)}[k,k_1] \wedge Formula[k_1,k_2] \wedge \\ & \quad \wedge PATTERN_{\theta(\prime\to(\prime)}[k_2,k_3] \;\wedge\; Formula[k_3,k_4] \wedge \\ & \quad \wedge PATTERN_{\theta(\prime\to\prime)}[k_4,k_5] \wedge segeqseg[k_5,k_6,k_1,k_2] \wedge \\ & \quad \wedge PATTERN_{\theta(\prime))\prime)}[k_6,n]). \end{aligned}$$

We also will need boolean expressions $TermSubstitution[k,n,k_t,n_t,k_v,n_v,k',n']$ and $FormulaSubstitution[k,n,k_t,n_t,k_v,n_v,k',n']$. The first of these expressions states that the term $p^*_k, \ldots, p^*_{n-1}$ is the result of substitution of the term $p^*_{k_t}, \ldots, p^*_{n_t-1}$ for the variable $p^*_{k_v}, \ldots, p^*_{n_v-1}$ into the term $p^*_{k'}, \ldots, p^*_{n'-1}$. The second expression states that a predicate formula $p^*_k, \ldots, p^*_{n-1}$ is the result of substitution of a term $p^*_{k_t}, \ldots, p^*_{n_t-1}$ for all free occurrences of the variable $p^*_{k_v}, \ldots, p^*_{n_v-1}$ into the predicate formula $p^*_{k'}, \ldots, p^*_{n'-1}$. Such boolean expressions can be specified through recursion over parameters k and n in a way which is similar to the recursive definitions of $Term[k,n]$ and $Formula[k,n]$. The only new element of such definitions would be the use of the expression $segeqseg[i,j,i',j']$ to state that certain segments of the sequence $p^*_0, p^*_1, \ldots$ are equal. Since the use of the expression $segeqseg[i,j,i',j']$ has already been illustrated in the above definition of the formula $A[k,n]$, we will spare the details.

Using the expression $FormulaSubstitution[k,n,k_t,n_t,k_v,n_v,k',n']$, one can write boolean expressions for the propositional axioms of Peano Arithmetic, such as $A[k,n]$ above. In addition, one can also write similar boolean expressions for first-order axiom schemata and arithmetic-specific axioms (as well as axiom schemata). Since the total number of axiom schemata and stand-alone axioms is finite, we can write a propositional formula $Axiom[k,n]$ that says that a binary string $p^*_k, \ldots, p^*_{n-1}$ is equal to a translation of some axiom of arithmetic under encoding θ.

Next, we will define a propositional formula $Proof[n,s]$ such that for any non-negative integers n and s and any valuation *, the boolean expression $Proof[n,s]$ is true if and only if the boolean string $p^*_0, p^*_1, \ldots, p^*_{n-1}$ is a translation, under encoding θ, of a Peano Arithmetic proof that contains exactly s formulas.

Definition 14 *For any non-negative integers n and s, the recursive boolean*

expression $Proof[n,s]$ *is defined as follows:*

$$\begin{aligned}
Proof[0,0] &= \top,\\
Proof[n+1,0] &= \bot,\\
Proof[0,s+1] &= \bot,\\
Proof[n+1,s+1] &= \bigvee_{k_1=0}^{n}\bigvee_{k_2=0}^{n}(Proof[k_1,s-1]\wedge\\
&\qquad \wedge PATTERN_{\theta(',')}[k_1,k_2]\wedge Axiom[k_2,n]) \quad \vee\\
&\qquad \bigvee_{i_1=0}^{n}\bigvee_{i_2=0}^{n}\bigvee_{i_3=0}^{n}\bigvee_{i_4=0}^{n}\bigvee_{j_1=0}^{n}\bigvee_{j_2=0}^{n}\bigvee_{j_3=0}^{n}\bigvee_{j_4=0}^{n}\bigvee_{j_5=0}^{n}\bigvee_{j_6=0}^{n}\bigvee_{k=0}^{n}\\
&\qquad (PATTERN_{\theta(',')}[i_1,i_2]\wedge Formula[i_2,i_3]\wedge\\
&\qquad \wedge PATTERN_{\theta(',')}[i_3,i_4]\wedge\\
&\qquad \wedge (lesseq[i_4,j_1]\vee lesseq[j_6,i_1])\wedge\\
&\qquad \wedge PATTERN_{\theta(',(')}[j_1,j_2]\wedge\\
&\qquad \wedge segeqseg[i_2,i_3,j_2,j_3]\wedge\\
&\qquad \wedge PATTERN_{\theta('\to')}[j_3,j_4]\wedge\ Formula[j_4,j_5]\wedge\\
&\qquad \wedge PATTERN_{\theta(('),')}[j_5,j_6]\wedge\\
&\qquad \wedge lesseq[i_4,k]\wedge lesseq[j_6,k]\wedge\\
&\qquad \wedge Proof[k,s-1]\wedge\ segeqseg[j_4,j_5,k,n]).
\end{aligned}$$

3.6 Provability Formula $\Box_n\alpha$

Definition 15 *For any arithmetical formula* α*, let* $\Box_n\alpha$ *be the boolean expression*

$$\begin{aligned}
\Box_n\alpha &= \bigvee_{l=0}^{n}(\bigvee_{s=0}^{l} Proof[l,s])\wedge\\
&\qquad \wedge \bigvee_{k_1=0}^{l}\bigvee_{k_2=0}^{l}(PATTERN_{\theta(',')}[k_1,k_2]\wedge PATTERN_{\theta(\alpha)}[k_2,l])).
\end{aligned}$$

Lemma 15 *For any non-negative integers* n*, and any arithmetical formula* α*, the boolean expression* $\Box_n\alpha$ *is satisfiable if and only if the formula* α *has a proof in Peano Arithmetic of length no more than* n*.* □

Finally, let Δ_n^α be defined as $\neg\Box_{E(n)}\alpha$.

3.7 System $\mathcal{R}$

We are ready to define the system of boolean recursions $\mathcal{R}$. It is simply the system that can be obtained by combining recursive definitions of $eq[i,j]$, $less[i,j]$, $addeq[i,j,k]$, $prev[i,j]$, $subeqsub[i,j,i',j']$, $segeqseg[i,j,i',j']$, $Variable[k,n]$,

$Term[k,n]$, $Atom[k,n]$, $Formula[k,n]$, $TermSubstitution[k,n,k_t,n_t,k_v,n_v,k',n']$, $FormulaSubstitution[k,n,k_t,n_t,k_v,n_v,k',n']$, $Axiom[k,n]$, $Proof[n,s]$, and Δ_n^α given above.

Lemma 16 *There is a constants c_1, c_2 and c_3 such that*

$$|\Delta_l^\alpha|_{\mathcal{R}} \leq c_1 + c_2|\alpha| + c_3 l.$$

Proof. See Lemma 5. Constant c_1 is a function of $\mathcal{R}$. □

Lemma 17 *For any positive integer n and any arithmetical formula α,*

$$\vdash \Delta_n^\alpha \quad \Longrightarrow \quad PA \vdash \neg Pr_{e(n)}(\ulcorner \alpha \urcorner).$$

Proof. Consider any proof of $\neg\Box_{E(l)}\alpha$ in the propositional logic. Translate this proof into a proof in Peano Arithmetic by interpreting each propositional variable p_i as the arithmetical statement "i-th bit in the proof π is equal to 1". The result of the translation is a PA proof that any Peano Arithmetic proof π of length no longer than $e(n)$ does not prove the arithmetical formula α. □

Lemma 18 *There is a polynomial $p_2(x)$ such that for any positive integer n and any arithmetical statement α,*

$$PA \vdash_x Taut(\ulcorner \Delta_n^\alpha \urcorner) \quad \Longrightarrow \quad PA \vdash_{p_2(x)} Pr(\ulcorner \neg Pr_{e(n)}(\ulcorner \alpha \urcorner) \urcorner).$$

Proof. By formalization of proof of Lemma 17. □

4 Arithmetical Lemmas

Below we reproduce the standard diagonalization lemma for Peano Arithmetic. We only add polynomial bound on the proof size. The standard proof of the diagonalization lemma, also reproduced below, works in this case.

Lemma 19 *There is a monotonic polynomial $p_3(n)$ such that for any arithmetical formula $\alpha(x)$ with a single free variable x, there is an arithmetical statement ϕ such that*

$$PA \vdash_{p_3(|\phi|)} \phi \leftrightarrow \alpha(\ulcorner \phi \urcorner).$$

Proof. Let $sub(a,b)$ be a term that takes a Gödel number a of an arithmetical formula with a single free variable x and a Gödel number b of another arithmetical formula and computes the Gödel number of the result of substitution of the second formula for the variable x into the first formula. Let k be the Gödel number of formula $\alpha(sub(x,x))$. Define the statement ϕ to be $\alpha(sub(k,k))$. Note that

$$\alpha(\ulcorner \phi \urcorner) = \alpha(\ulcorner \alpha(sub(k,k)) \urcorner) = \alpha(sub(\ulcorner \alpha(sub(x,x)) \urcorner, k)) = \alpha(sub(k,k)) = \phi.$$

The last equality can be established inside Peano Arithmetic as an equality of two Gödel numbers: $PA \vdash \ulcorner \alpha(\ulcorner \phi \urcorner) \urcorner = \ulcorner \phi \urcorner$. It will be done by structural induction on subformulas of ϕ. Thus, the entire proof is bounded in size by a polynomial function of $|\phi|$. Finally, by one of the axioms of equality, $PA \vdash \ulcorner \alpha(\ulcorner \phi \urcorner) \urcorner = \ulcorner \phi \urcorner$ implies that $PA \vdash \alpha(\ulcorner \alpha(\ulcorner \phi \urcorner) \urcorner) \leftrightarrow \alpha(\ulcorner \phi \urcorner)$. Therefore, $PA \vdash \phi \leftrightarrow \alpha(\ulcorner \phi \urcorner)$, and it can be established in Peano Arithmetic by a proof whose size is bounded by a polynomial function of $|\phi|$. □

Lemma 20 *There is a monotonic polynomial $q(x)$ such that if $PA \vdash_n \phi$, then $PA \vdash_{q(n)} Pr_n(\ulcorner \phi \urcorner)$.* □

Lemma 21 *There is a monotonic polynomial $p_4(x, y)$ such that if $PA \vdash_n \phi \leftrightarrow \psi$ and $PA \vdash_m Pr(\ulcorner \psi \urcorner)$, then $PA \vdash_{p_4(n,m)} Pr_n(\ulcorner \phi \urcorner)$.*

Proof. Follows from the previous lemma. □

5 Arithmetical Statement ω

In this section we will use the diagonalization lemma to define a provable arithmetical statement ω such that formula $Pr(\ulcorner \omega \urcorner)$ has no short arithmetical proofs.

Recall that $f \prec e$. Let N be such that for any $n > N$

$$p_4(p_3(n), p_2(f(c_1 + c_2 \cdot p_1(n) + c_3(n)))) < e(n) \tag{8}$$

By Lemma 19, there is an arithmetical statement ω such that

$$PA \vdash_{p_3(|\omega|)} \omega \leftrightarrow \neg Pr_{e(\lfloor \log \ulcorner \omega \urcorner \rfloor + 1 + N)}(\ulcorner Pr(\ulcorner \omega \urcorner) \urcorner).$$

Let N_ω be the size of formula ω. By Lemma 2, $N_\omega = \lfloor \log(\ulcorner \omega \urcorner) \rfloor + 1$. Hence,

$$PA \vdash_{p_3(N_\omega)} \omega \leftrightarrow \neg Pr_{e(N_\omega + N)}(\ulcorner Pr(\ulcorner \omega \urcorner) \urcorner). \tag{9}$$

Lemma 22 *The statement ω is true in the standard model of Peano Arithmetic.*

Proof. By contradiction. Assume that the statement ω is false. Thus, by (9), the statement $Pr_{e(N_\omega + N)}(Pr(\ulcorner \omega \urcorner))$ is true. Hence, the statement $Pr(\ulcorner \omega \urcorner)$ is true. Therefore, the statement ω is true. Contradiction. □

Lemma 23 *The statement ω is provable in Peano Arithmetic.*

Proof. This is a true statement which, by (9), is provably equivalent to a statement that contains only bounded quantifiers. □

Lemma 24 $PA \nvdash_{e(N_\omega + N)} Pr(\ulcorner \omega \urcorner)$.

Proof. By Lemma 22, the statement ω is true. Thus, by the equivalence (9), the statement $Pr_{e(N_\omega+N)}(Pr(\ulcorner\omega\urcorner))$ is false. Hence, $PA \nvdash_{e(N_\omega+N)} Pr(\ulcorner\omega\urcorner)$. $\square$

Therefore, ω is an explicit example of a hard-to-prove arithmetical statement. In the next section we will use the statement ω to build a hard-to-prove propositional formula and, thus, prove Theorem 1.

6 Proof of Theorem 1

Proof. Let $\mathcal{R}$ be the defined in Section 3.7 system of boolean recursions. Let b be boolean expression $\Delta_{N_\omega+N}^{Pr(\ulcorner\omega\urcorner)}$. We will show that the formula b is a tautology such that $||b||_{PA} > f(|b|_{\mathcal{R}})$.

First, we will show that formula b is a tautology. Assume the opposite. Thus, the propositional formula $\Box_{e(N_\omega+N)}Pr(\ulcorner\omega\urcorner)$ is satisfiable. Hence, by Lemma 15, $PA \vdash_{e(N_\omega+N)} Pr(\ulcorner\omega\urcorner)$. Contradiction with Lemma 24.

Next, we will show that $PA \nvdash_{f(|b|_{\mathcal{R}})} Taut(\ulcorner b\urcorner)$. Assume the opposite. Thus, by the definition of b,

$$PA \vdash_{f(|b|_{\mathcal{R}})} Taut(\ulcorner\Delta_{N_\omega+N}^{Pr(\ulcorner\omega\urcorner)}\urcorner).$$

By Lemma 18,

$$PA \vdash_{p_2(f(|b|_{\mathcal{R}}))} Pr(\ulcorner\neg Pr_{e(N_\omega+N)}(\ulcorner Pr(\ulcorner\omega\urcorner)\urcorner)\urcorner).$$

By Lemma 21 and statement (9),

$$PA \vdash_{p_4(p_3(N_\omega),p_2(f(|b|_{\mathcal{R}})))} Pr(\ulcorner\omega\urcorner).$$

By Lemma 16 and the monotonicity of p_4, p_2, and f,

$$PA \vdash_{p_4(p_3(N_\omega),p_2(f(c_1+c_2\cdot|Pr(\ulcorner\omega\urcorner)|+c_3(N_\omega+N))))} Pr(\ulcorner\omega\urcorner).$$

By Lemma 4 and the monotonicity of p_2, p_4, and p,

$$PA \vdash_{p_4(p_3(N_\omega),p_2(f(c_1+c_2\cdot p_1(N_\omega)+c_3(N_\omega+N))))} Pr(\ulcorner\omega\urcorner).$$

By monotonicity of f, p_1, p_2, p_3, and p_4,

$$PA \vdash_{p_4(p_3(N_\omega+N),p_2(f(c_1+c_2\cdot p_1(N_\omega+N)+c_3(N_\omega+N))))} Pr(\ulcorner\omega\urcorner).$$

By the choice of N, see inequality (8),

$$PA \vdash_{e(N_\omega+N)} Pr(\ulcorner\omega\urcorner).$$

Contradiction with Lemma 24. $\square$

7 Acknowledgments

The author would like to thank Klaus Aehlig and Jan Krajíček for the discussion of this work and the constructive comments they gave. A special credit should be given to the anonymous reviewers whose suggestions led to more a general and clean statement of the main result. All remaining errors are solely the author's responsibility.

References

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

[2] W. Cook, C. R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Appl. Math.*, 18(1):25–38, 1987.

[3] Armin Haken. The intractability of resolution. *Theoret. Comput. Sci.*, 39(2-3):297–308, 1985.

[4] Jan Krajíček. Diagonalization in proof complexity. *Fund. Math.*, 182(2):181–192, 2004.

[5] Jan Krajíček. Implicit proofs. *J. Symbolic Logic*, 69(2):387–397, 2004.

[6] Jan Krajíček. Structured pigeonhole principle, search problems and hard tautologies. *J. Symbolic Logic*, 70(2):619–630, 2005.

[7] Jan Krajíček, Pavel Pudlák, and Alan Woods. An exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Structures Algorithms*, 7(1):15–39, 1995.

[8] R. J. Parikh. Some results on the lengths of proofs. *Transactions of the American Mathematical Society*, 177:29–36, 1973.

[9] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Comput. Complexity*, 3(2):97–140, 1993.