Propositional Consistency Proofs

Samuel R. Buss*
Department of Mathematics
University of California, San Diego

July 11, 2002

Abstract

Partial consistency statements can be expressed as polynomial-size propositional formulas. Frege proof systems have polynomial-size partial self-consistency proofs. Frege proof systems have polynomial-size proofs of partial consistency of extended Frege proof systems if and only if Frege proof systems polynomially simulate extended Frege proof systems. We give a new proof of Reckhow's theorem that any two Frege proof systems p-simulate each other.

The proofs depend on polynomial size propositional formulas defining the truth of propositional formulas. These are already known to exist since the Boolean formula value problem is in alternating logarithmic time; this paper presents a proof of this fact based on a construction which is somewhat simpler than the prior proofs of Buss and of Buss-Cook-Gupta-Ramachandran.

1 Introduction

It is a celebrated result of Gödel that a sufficiently strong, consistent theory can not prove its own consistency. Consider, however, a *partial* consistency statement such as $Con_{ZF}(n)$ which states that there is no ZF-proof of a

*Supported in part by NSF Grants DMS-8701828 and DMS-8902480.

contradiction with length $\leq n$ symbols (i.e., with a total of n or fewer symbols). As a true primitive recursive property, $Con_{ZF}(n)$ is provable in ZF for each particular value of $n \geq 0$, even though $(\forall x)Con_{ZF}(x)$ is not a consequence of ZF. Indeed, for each n, $Con_{ZF}(n)$ is provable in weak fragments of arithmetic such as $I\Delta_0$ or S^1_2, or even Q. A natural question is how long or complex such proofs are. Friedman (unpublished) and, independently, Pudlák [8, 9] partially answered this by showing there is a polynomial $p(n)$ such that any ZF-proof of $Con_{ZF}(p(n))$ requires length at least n and such that $Con_{ZF}(n)$ has a ZF-proof of $\leq p(n)$ symbols. This result seems to generalize easily to stronger theories and also applies to Peano arithmetic and even weaker fragments of arithmetic such as Bounded Arithmetic (in place of *all* uses of ZF). On the other hand, there is a close connection between results of this kind and the NP =?coNP question. Indeed, if a formal theory T can be found such that $Con_T(n)$ requires exponential size ZF-proofs then ZF does not prove that NP is closed under complementation (see Krajíček-Pudlák [7] for this and further connections to computational complexity).

It is interesting to inquire whether such results hold for substantially weaker systems in place of ZF. The two natural weak theories to consider are propositional proof systems: Frege proof systems (denoted $\mathcal{F}$) are the usual propositional proof systems with modus ponens as the only inference rule and extended Frege systems (denoted $e\mathcal{F}$) are Frege systems plus an additional *extension rule* which allows introduction of abbreviations. It is a somewhat surprising and initially counterintuitive fact that there are polynomial size propositional formulas for expressing partial consistency statements such as $Con_{\mathcal{F}}(n)$, $Con_{e\mathcal{F}}(n)$ and even $Con_{ZF}(n)$. To formulate these partial consistency statements as propositional formulas one must encode metamathematical (syntactic) concepts such as 'formula' and 'proof' as strings in the two character alphabet $\{\top, \bot\}$ and thereby let a sequence of propositional variables denote a formula or a proof. The statement $Con_{\mathcal{F}}(n)$ is expressed (in a manner to be made precise below) by saying that a sequence of $c \cdot n$ propositional variables does not code a proof of a contradiction (where c is an appropriate constant).

Cook [5] showed that there are polynomial size extended Frege proofs of the partial self-consistency statements $Con_{e\mathcal{F}}(n)$. This was the first such result, predating the above-mentioned results for Peano arithmetic and set theory.

This paper proves that Frege systems also have polynomial size proofs of partial self-consistency. Our proof depends critically on the fact that the Boolean formula value problem is in alternating logarithmic time [1, 3], or more precisely, on the fact that there are polynomial size propositional formulas which define the truth value of propositional formulas. In section 3 below, we reprove this fact using a simpler construction than was employed in the prior proofs in [1, 3].[1] Our proof method also gives a new proof of Reckhow's theorem that any two Frege systems p-simulate each other [10]. We also show that Frege systems simulate extended Frege systems if and only if there are polynomial size Frege proofs of $Con_{e\mathcal{F}}(n)$.

We begin by reviewing Frege and extended Frege proof systems: for a more detailed treatment see [10, 6]; some of the proofs may also be found in [4]. A *Frege system* $\mathcal{F}$ is a proof system for propositional formulas in a language $\mathcal{L}$ and has a finite set of axiom schemes. The language $\mathcal{L}$ consists of a finite, truth-functionally complete set of propositional connectives; the axiom schemes are tautologies, for example, $\varphi \rightarrow (\psi \rightarrow \varphi)$ where φ and ψ may be arbitrary formulas. The proof system $\mathcal{F}$ has modus ponens as its only rule of inference and must be complete. An *extended Frege system* $e\mathcal{F}$ is defined similarly but has an additional rule of inference called the *extension rule* which allows introduction of abbreviations; the extension rule allows the derivation of $p \leftrightarrow \varphi$ where p is a new variable which has not been used yet in the proof and does not appear in φ or in the final line (the proved formula) of the proof.

The *length* or *size* of a proof is the total number of symbols appearing in the proof. It is easy to see that the particular choice of axiom schemes for a Frege or extended Frege system will alter proof lengths by only a constant factor (for this it is crucial that there is only a *finite* number of axiom schemes); so, for us, the precise choice of axiom schemes is unimportant. However, the choice of language $\mathcal{L}$ is more problematic (but see below).

A *proof system* is defined to be a polynomial time function f from $\{0,1\}^*$ *onto* the set of tautologies in some propositional language. Frege and extended

[1] Actually, the original motivation for the discovery of the alternating log time algorithm for the Boolean formula value problem was to obtain the results of this paper. The prior constructions of [1, 3] could also be used to give the polynomial size Frege proofs of partial self-consistency; however, the simpler construction of this paper substantially reduces (by about two-thirds) the number of cases that must be considered in the definition of $SubFm_k$ below.

Frege systems can be viewed as proof systems in this sense by letting $f(w)$ be the final line of w if w codes a valid proof and be some arbitrary tautology otherwise.

Suppose f_1 and f_2 are proof systems over languages $\mathcal{L}_1$ and $\mathcal{L}_2$. We say that f_1 *simulates* f_2 if there are functions g and h of polynomial growth rate such that for all $\mathcal{L}_1$-formulas φ, $g(\varphi)$ is a $\mathcal{L}_2$-formula and $f_1(h(w,\varphi)) = \varphi$ whenever $f_2(w) = g(\varphi)$. The idea is that g translates φ into the language $\mathcal{L}_2$ and h translates any f_2-proof of $g(\varphi)$ into an f_1-proof of φ. If g and h are polynomial time computable then we say f_1 *p-simulates* f_2. Sometimes additional constraints are put on g; in particular, $g(\varphi)$ is sometimes expected to be tautologically equivalent to φ and if $\mathcal{L}_1 \subseteq \mathcal{L}_2$ then g might be required to be the identity function.

A fundamental result, due to Reckhow [10] is that any two Frege systems p-simulate each other and any two extended Frege systems p-simulate each other. Let $\mathcal{F}_0$ be a Frege proof system with language $\{\neg, \wedge\}$ and let $\mathcal{F}_+$ be a Frege proof system whose language contains all thirteen nullary, unary and binary propositional connectives (not counting connectives which do not depend on all their arguments). The difficult part of Reckhow's theorem on Frege systems is showing that $\mathcal{F}_0$ p-simulates $\mathcal{F}_+$ with the identity translation function g. In the setting of extended Frege systems, this direction can be handled in a high level way: let $e\mathcal{F}_0$ and $e\mathcal{F}_+$ be the above systems augmented with the extension rule; Cook has shown that $e\mathcal{F}_0$ can p-simulate any proof system that PV (or, equivalently, S^1_2) proves consistent; so in particular, $e\mathcal{F}_0$ p-simulates $e\mathcal{F}_+$.

If $\varphi_1, \varphi_2, \varphi_3, \ldots$ is a family of formulas we write $T \vdash^* \varphi_n$ to mean that the proof system T has polynomial size proofs of the φ_n's; i.e., that there is a polynomial p such that for all n there is a T-proof of φ_n of size $p(|\varphi_n|)$ where $|\varphi_n|$ is the length of φ_n.

If T is a proof system, we let $Con_T(n)$ be a propositional formula that expresses the fact that there is no T-proof of length $\leq n$ of some false formula ($p \wedge \neg p$, say). The formula $Con_T(n)$ is more fully defined in the next section; an important property is that the length of $Con_T(n)$ is bounded by a polynomial of n.

This paper proves the following results:

Main Theorem 1 *$\mathcal{F}_0 \vdash^* Con_{\mathcal{F}_+}(n)$. More generally, if $\mathcal{F}_1$ and $\mathcal{F}_2$ are Frege proof systems, then $\mathcal{F}_1 \vdash^* Con_{\mathcal{F}_2}(n)$.*

Main Theorem 2 *(Reckhow [10]). $\mathcal{F}_0$ p-simulates $\mathcal{F}_+$. More generally, for any two Frege proof systems $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1$ p-simulates $\mathcal{F}_2$.*

Main Theorem 3 *Frege proof systems p-simulate extended Frege proof systems if and only if $\mathcal{F} \vdash^* Con_{e\mathcal{F}}(n)$ (for $\mathcal{F}$ any Frege proof system).*

Theorem 1 will be proved by giving a polynomial size propositional truth definition for propositional formulas. The existence of such a polynomial size truth definition is roughly equivalent to the existence of an alternating logarithmic time algorithm for recognizing true propositional sentences. Theorems 2 and 3 will be corollaries of the method of proof of Theorem 1. Reckhow's original proof of Theorem 2 used the Spira method of evaluating Boolean formulas; our proof can be viewed as a more sophisticated version of Reckhow's proof.

2 Formalizing Metamathematics in Frege Systems

2.1 Integers, Symbols, Formulas and Proofs

This section discusses how to formalize metamathematics, especially of propositional proof systems, inside a Frege proof system $\mathcal{F}$. For notational convenience, the system $\mathcal{F}$ will use the language $\{\neg, \wedge, \vee, \rightarrow\}$; it is easy to see that $\mathcal{F}_0$ p-simulates $\mathcal{F}$ since $\vee$ and $\rightarrow$ can be efficiently expressed with $\neg$ and $\wedge$; by "efficiently expressed" we mean that the natural direct translation of formulas involving $\vee$ and $\rightarrow$ to ones involving only $\neg$ and $\wedge$ only increases the formula size by a constant factor. Hence our results apply to $\mathcal{F}_0$ as well.

We begin with an explanation of the notation for propositional formulas. The expression $\varphi \leftrightarrow \psi$ is an abbreviation for the $\mathcal{F}_0$-formula $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$; whereas the symbol $\equiv$ denotes the binary biconditional connective (in $\mathcal{F}_+$, for example). Formally speaking, a Frege proof system has propositional variables $p_0, p_1, p_2, \ldots$; we also use $s, x, y, \ldots$ with sub- and superscripts as metasymbols for propositional variables. Propositional

formulas are always fully parenthesized to indicate precedence, although we frequently do not display all the parentheses. Greek letters $\varphi, \psi, \ldots$ denote propositional formulas. Symbols $\bigwedge\!\!\bigwedge$ and $\bigvee\!\!\bigvee$ denote conjunction and disjunction of a set of formulas; since we are only interested in polynomial bounds on proof size the precise method of associating $\wedge$'s and $\vee$'s is unimportant as arbitrary regrouping with commutative and associative laws can always be done with polynomial size proofs.

Lemma 4 *(Length Minimization Lemma)*

$$\mathcal{F} \vdash^* \bigvee_{i=1}^{n} \varphi_i \rightarrow \bigvee_{i=1}^{n} \left(\varphi_i \wedge \bigwedge_{j=1}^{i-1} \neg \varphi_j \right)$$

Proof This is very simple: $\mathcal{F}$ just proves the formula successively for $n = 1, 2, \ldots$. $\square$

Although Frege proofs systems only deal with variables that range over *True* and *False*, it is possible to indirectly deal with integers by coding an integer n in binary notation. If $n < 2^j$ then the j variables $x_{j-1}, \ldots, x_0$ can be used to represent n by letting x_i be true if and only if the i-th bit of the binary representation of n is a 1. In effect this allows us to conservatively extend a Frege system to a two-sorted theory with propositional and integer "sorts". Accordingly, we will let $I, J, K, \ldots$ denote integers which are introduced in a Frege proof by being bit-wise coded. It is also possible to define $I + J$, $I = J$, $I \leq J$, $I \dot{-} J$, $\max(I, J)$, etc.,[2] in a Frege system; more precisely, given vectors of propositional formulas which define the binary representations of I and J, it is possible to express the binary representation of $I + J$, etc., by a vector of polynomial size propositional formulas. (Here, polynomial size means polynomial in the size of the formulas representing I and J.) For example, if x_1, x_0 and y_1, y_0 are propositional variables that code two natural number < 4, then their sum can be defined by the vector of formulas $\varphi_2, \varphi_1, \varphi_0$:

$$\begin{aligned} \varphi_0 &\iff^{\text{df}} x_0 \leftrightarrow \neg y_0 \\ \varphi_1 &\iff^{\text{df}} (x_1 \leftrightarrow y_1) \leftrightarrow (x_0 \wedge y_0) \\ \varphi_2 &\iff^{\text{df}} (x_1 \wedge y_1) \vee (x_0 \wedge y_0 \wedge (x_1 \vee y_1)) \end{aligned}$$

[2]The $\dot{-}$ means restricted subtraction; i.e., $I \dot{-} J$ is equal to the maximum of zero and $I - J$.

See Buss [2] for an exposition of the details of handling integers in Frege system and for proofs that simple properties of $+$, $\leq$, etc have polynomial size proofs. In particular, in [2], we proved the existence of polynomial size formulas for counting:

Lemma 5 *The integers*

$$I_{j,k} \;=\; \text{the number of true } x_n \text{ with } j \leq n \leq k$$

can be defined with polynomial size formulas in the Frege system $\mathcal{F}$. Furthermore

$$\mathcal{F} \vdash^* (x_j \rightarrow I_{j,j} = 1) \wedge (\neg x_j \rightarrow I_{j,j} = 0)$$

$$\mathcal{F} \vdash^* I_{j,k} + I_{k+1,\ell} = I_{j,\ell}$$

The proof of Lemma 5 in [2] uses a technique known as carry-save-addition to define $I_{j,k}$; the same methods are also used to show that multiplication and vector summation are definable by Frege proof systems. As a shorthand notation we will use '#' as a symbol for 'the number of'; as in,

$$I_{j,k} = (\#n, j \leq n \leq k)(x_n)$$

.

Proofs in the Frege system $\mathcal{F}_+$ will be represented by words in the 19 character alphabet Σ containing p, 0, 1, parentheses, comma and 13 propositional connectives. A propositional variable p_i will be represented by "p" followed by a string of 0's and 1's coding i in binary. Commas are used to separate formulas in a proof. Strings over Σ are further encoded in the language $\{\top, \bot\}$ by assigning a unique 5-bit code to each symbol in Σ. Thus an $\mathcal{F}_+$ formula with k symbols (counting the symbols used to code the subscripts of variables) will be coded by $5k$ truth values.

Let a boldface $\mathbf{x}$ represent a vector of propositional variables $x_1, \ldots, x_{5k}$. We want to define concepts such as "$\mathbf{x}$ codes an $\mathcal{F}_+$-formula", "$\mathbf{x}$ codes an $\mathcal{F}_+$-proof", etc. Let $Sym_i^{\mathbf{x}}$ denote the i-th symbol from Σ in $\mathbf{x}$; namely, the symbol coded by $x_{5i-4}, \ldots, x_{5i}$. A *logical symbol* is a parenthesis, comma, propositional connective or propositional variable, and in the last case is coded by more than one Σ-symbol. Let $\mathbf{x}[\mathbf{i}]$ denote the i-th logical symbol in $\mathbf{x}$. Let $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ be the substring of $\mathbf{x}$ from $\mathbf{x}[\mathbf{i}]$ through $\mathbf{x}[\mathbf{j}]$ inclusive. There are polynomial size propositional formulas for manipulating $\mathbf{x}[\mathbf{i}]$ and $\mathbf{x}[\mathbf{i}, \mathbf{j}]$; for example, define:

$$Sym_j^{\mathbf{x}} \text{ is in } \mathbf{x}[\mathbf{i}] \quad \stackrel{\text{df}}{\iff} \quad i = (\#k \leq j)[Sym_k^{\mathbf{x}} \text{ is not 0 or 1}]$$

and

$$Sym_j^{\mathbf{x}[\mathbf{i}]} = Sym_k^{\mathbf{x}} \quad \text{where } j = (\#\ell \leq k)[Sym_\ell^{\mathbf{x}} \text{ is in } \mathbf{x}[\mathbf{i}]].$$

Note that logical symbols in $\mathbf{x}$ can be counted merely by counting the Σ-symbols other than 0 and 1. Hence the definition of "$Sym_j^{\mathbf{x}}$ is in $\mathbf{x}[\mathbf{i}]$" can be written as a polynomial size Boolean formula by Lemma 5, and similarly, $Sym_j^{\mathbf{x}[\mathbf{i}]}$ is a vector of five polynomial size formulas. The free variables in these formulas are the variables $\mathbf{x}$ and the variables encoding the integers i and j.

The point of the above is that the Frege system $\mathcal{F}$ can handle concepts such as Σ-symbols, logical symbols, $Sym_i^{\mathbf{x}}$, $\mathbf{x}[\mathbf{i}]$ and $\mathbf{x}[\mathbf{i},\mathbf{j}]$. It is convenient to informally view the system $\mathcal{F}$ as being conservatively extended with new "sorts" for these concepts just as integers coded in binary can be viewed as a sort.

The key tool for parsing a formula coded by $\mathbf{x}$ is counting parentheses. Let $|\mathbf{x}|$ denote the number of logical symbols in $\mathbf{x}$ and let $|\mathbf{x}|_\Sigma$ denote the number of Σ-symbols in $\mathbf{x}$. Note that if $\mathbf{x}$ codes a formula φ then $|\mathbf{x}| = |\varphi|$, and if every variable has subscript of length $\leq i$ then $|\mathbf{x}|_\Sigma \leq (i+1) \cdot |\varphi|$. We define:

$$\mathbf{x} \textit{ is balanced} \stackrel{\text{df}}{\iff} (\#i \leq |\mathbf{x}|)\Big[\mathbf{x}[\mathbf{i}] = \text{`('}\Big] = (\#i \leq |\mathbf{x}|)\Big[\mathbf{x}[\mathbf{i}] = \text{`)'}\Big]$$
$$\text{and} \bigwedge\!\!\!\bigwedge_{k=1}^{|\mathbf{x}|-1} \Big((\#i \leq k)\Big[\mathbf{x}[\mathbf{i}] = \text{`('}\Big] > (\#i \leq k)\Big[\mathbf{x}[\mathbf{i}] = \text{`)'}\Big]\Big)$$

$$\mathbf{x} \textit{ codes a constant} \stackrel{\text{df}}{\iff} |\mathbf{x}| = 1 \text{ and } \mathbf{x}[\mathbf{1}] \text{ is } \top \text{ or } \bot$$

$$\mathbf{x} \textit{ codes a variable} \stackrel{\text{df}}{\iff} Sym_1^{\mathbf{x}} = \text{`}p\text{'} \wedge (\forall j, 1 < j \leq |\mathbf{x}|_\Sigma)(Sym_j^{\mathbf{x}} \text{ is 0 or 1})$$

$$\mathbf{x} \textit{ codes an atomic formula} \stackrel{\text{df}}{\iff} \mathbf{x} \text{ codes a variable or a constant}$$

$$\mathbf{x} \textit{ is fmla-like} \stackrel{\text{df}}{\iff} \mathbf{x} \text{ is balanced or codes an atomic formula}$$

$$\begin{aligned}\mathbf{x}\ \textit{codes a formula} \iff^{\text{df}}\ & \mathbf{x} \text{ is fmla-like and} \\ & (\forall i)\Big(\mathbf{x}[\mathbf{i}] = \text{`(`} \to (\exists j)\Big(\mathbf{x}[\mathbf{i},\mathbf{j}] \text{ is fmla-like and} \\ & \quad \Big[(\mathbf{x}[\mathbf{i}+\mathbf{1}] \text{ is a unary connective and } \mathbf{x}[\mathbf{i}+\mathbf{2},\mathbf{j}-\mathbf{1}] \text{ is fmla-like}) \\ & \quad \text{or } (\exists k)(i < k < j \wedge \mathbf{x}[\mathbf{k}] \text{ is a binary connective and} \\ & \qquad \mathbf{x}[\mathbf{i}+\mathbf{1},\mathbf{k}-\mathbf{1}] \text{ and } \mathbf{x}[\mathbf{k}+\mathbf{1},\mathbf{j}-\mathbf{1}] \text{ are fmla-like})\Big]\Big)\Big)\end{aligned}$$

We have tried to phrase the definition of "$\mathbf{x}$ is a formula" in a readable manner; but the crucial point is that the definition can be written as a polynomial size, propositional formula. Quantifiers such as $(\forall i)(\cdots)$ are to be read as $\bigwedge\!\!\!\bigwedge_{i=1}^{n}(i \leq |\mathbf{x}| \to \cdots)$ where $n = |\mathbf{x}|_\Sigma$. Furthermore, these formulas provide "intensional" definitions in that the Frege system $\mathcal{F}$ can prove simple propositions about propositional formulas. In particular, we have:

Lemma 6 *(Unique Readability) Suppose $i_1 \leq j_1 < i_2 \leq j_2$ with $i_1 \neq j_1$ or $i_2 \neq j_2$. Then*

$$\mathcal{F} \vdash^{*} \text{"At least one of } \mathbf{x}[\mathbf{i_1},\mathbf{i_2}] \text{ or } \mathbf{x}[\mathbf{j_1},\mathbf{j_2}] \text{ is not a formula."}$$

This lemma asserts that no two formulas can overlap unless one is a subformula of the other.

Proof We give an informal argument which may be formalized with polynomial size $\mathcal{F}$-proofs by the methods of Lemma 5. Let's suppose $i_1 < j_1$ and $i_2 < j_2$. Since $\mathbf{x}[\mathbf{j_1},\mathbf{j_2}]$ codes a formula, there are more left than right parentheses in $\mathbf{x}[\mathbf{j_1},\mathbf{i_2}]$. And likewise, there are equal numbers of left and right parentheses in $\mathbf{x}[\mathbf{i_1},\mathbf{i_2}]$ and more left than right parentheses in $\mathbf{x}[\mathbf{i_1},\mathbf{j_1}-\mathbf{1}]$. Hence there are more right than left parentheses in $\mathbf{x}[\mathbf{j_1},\mathbf{i_2}]$ which is a contradiction. The rest of the cases are similar. □

Propositional formulas can also be written in postfix (reverse Polish) notation. A PLOF formula (Postfix-Longer-Operands-First) is defined to be a postfix formula in which any subformula of the form $\varphi_1\varphi_2\cdots\varphi_k\circledcirc$, with $\circledcirc$ a k-ary connective and each φ_i a formula, has $|\varphi_i| \geq |\varphi_{i+1}|$ for all i. This concept can be defined with polynomial size formulas as follows (assume for the sequel that the language of $\mathbf{x}$ contains only logical connectives of arity less than or equal to 2; so in a PLOF formula, a binary connective never has its right operand longer than its left operand):

$$\begin{aligned}\mathbf{x}\ \textit{is a postfix formula} \iff^{\text{df}}\ & \mathbf{x}[\mathbf{1}] \text{ is an atomic formula and} \\ & (\#i)(\mathbf{x}[\mathbf{i}] \text{ is an atomic formula}) \\ & \qquad = 1 + (\#i)(\mathbf{x}[\mathbf{i}] \text{ is an binary connective}) \text{ and} \\ & (\forall j < |\mathbf{x}|)\Big[(\#i \le j)(\mathbf{x}[\mathbf{i}] \text{ is an atomic formula}) \\ & \qquad > (\#i \le j)(\mathbf{x}[\mathbf{i}] \text{ is an binary connective})\Big]\end{aligned}$$

$$\begin{aligned}\mathbf{x}\ \textit{is a PLOF formula} \iff^{\text{df}}\ & \mathbf{x} \text{ is a postfix formula and} \\ & \neg(\exists i < j < k)\Big[j - i < k - j \text{ and } \mathbf{x}[\mathbf{k}] \text{ is a binary connective} \\ & \qquad \text{and } \mathbf{x}[\mathbf{i}, \mathbf{j} - \mathbf{1}] \text{ and } \mathbf{x}[\mathbf{j}, \mathbf{k} - \mathbf{1}] \text{ are postfix formulas.}]\end{aligned}$$

Thus, polynomial size formulas for counting give polynomial size formulas for defining PLOF formulas. Note that the quantifier over i, j, k should be viewed as a disjunction over all appropriate values of i, j, k. It is straightforward to prove that the Unique Readability Lemma also applies to postfix formulas (provably in $\mathcal{F}$ with polynomial size proofs). And it is obvious that $\mathcal{F}$ has polynomial size proofs of the fact that any subformula of a PLOF formula is a PLOF formula.

Finally we need to define, via polynomial size formulas, the concept of a (Frege or extended Frege) proof. A proof is coded by formulas separated by commas where each formula is either an instance of an axiom scheme or follows by modus ponens from two previous formulas. For any fixed proof system, say $\mathcal{F}_+$, there are only a finite number of axiom schemes so it is easy to define the notion *"*$\mathbf{x}$ *is an instance of an axiom scheme"* with a polynomial size formula. Likewise, it is simple to define *"*$\mathbf{x}$ *is derived from* $\mathbf{y}$ *and* $\mathbf{z}$ *by modus ponens".* For infix formulas, modus ponens is defined as usual, provided implication ($\rightarrow$) is in the language; when $\rightarrow$ is not in the language then a tautologically equivalent formula is used in place of the assumption $A \rightarrow B$.

We also need to define the notion of (extended) Frege proofs for PLOF notation formulas. The main differences from proofs in infix notation are that the language has to be truth functionally complete with respect to PLOF formulas and that two distinct rules for modus ponens are required. Let PLOF-$\mathcal{F}_+$ be the proof system with all 13 nullary, unary and binary logical connectives and with formulas in PLOF notation; the modus ponens rules are:

$$\frac{AB \rightarrow \quad A}{B} \qquad \frac{AB \leftarrow \quad B}{A}$$

Here $\leftarrow$ is the reverse implication sign so $AB \rightarrow$ means $A \rightarrow B$ and $AB \leftarrow$ means $B \rightarrow A$. The reason two rules are needed is because the above rules are applicable only when the length of A is greater than or equal to the length of B.

Definition $Con_{\mathcal{F}}(\mathbf{x})$ is the polynomial size formula that says that $\mathbf{x} = x_1 \cdots x_{5n}$ does not code an $\mathcal{F}$-proof with final formula $p_0 \wedge \neg p_0$. $Con_{e\mathcal{F}}(\mathbf{x})$, $Con_{PLOF-\mathcal{F}_+}(\mathbf{x})$, etc are defined similarly.

Hence $Con_T(\mathbf{x})$ is a tautology if and only if there is no T-proof of a contradiction of $\leq n$ Σ-symbols (counting commas separating formulas and symbols in subscripts). Recall that $Con_T(n)$ meant that T-proofs with $\leq n$ logical symbols are consistent; so $Con_T(n)$ certainly implies $Con_T(\mathbf{x})$ is valid. Conversely, for Frege or extended Frege systems, if $Con_T(\mathbf{x})$ is valid then $Con_T(\alpha n)$ for some contant α which depends on T.[3] To see this, observe that if there is an inconsistency in T then the shortest contradiction can be presumed to use only the propositional variable p_0 (plus propositional variables which appear in axiom schemes); with this observation, it is easy to obtain α so that if there is a proof of a contradiction of k logical symbols then there is a proof coded with $\leq k/\alpha$ Σ-symbols.

2.2 Converting from Infix to PLOF

It is elementary that an infix notation formula can be converted into an equivalent PLOF formula—we show below that the propositional proof system $\mathcal{F}_0$ can describe this conversion. More precisely, given an infix formula coded by a sequence $\mathbf{x}$ of propositional variables, there is a sequence φ of polynomial size formulas which defines the natural equivalent PLOF formula. Of course the length of the PLOF formula coded by φ must also be defined by polynomial size formulas; this is easily done as the length is equal to the number of non-parenthesis Σ-symbols in the infix formula $\mathbf{x}$.

The equivalent PLOF formula is obtained by reordering the logical symbols of the infix formula and discarding parentheses. To do this we define when $\mathbf{x}[\mathbf{i}]$ is *before* $\mathbf{x}[\mathbf{j}]$ in the PLOF formula:

[3]However, usually $\alpha = \frac{1}{3}$ suffices; in particular, if every substitution instance of an axiom is an axiom.

$\mathbf{x}[\mathbf{i}]$ *is in the scope of* $\mathbf{x}[\mathbf{j}]$ $\stackrel{\text{df}}{\iff}$

$\Big[\mathbf{x}[\mathbf{j}]$ is a unary connective and

$(\exists k)[\mathbf{x}[\mathbf{j}+\mathbf{1},\mathbf{k}]$ codes a formula and $j < i \leq k]\Big]$ or

$\Big[\mathbf{x}[\mathbf{j}]$ is a binary connective and

$\Big((\exists k)(\mathbf{x}[\mathbf{j}+\mathbf{1},\mathbf{k}]$ codes a formula and $j < i \leq k)$

or $(\exists k)(\mathbf{x}[\mathbf{k},\mathbf{j}-\mathbf{1}]$ codes a formula and $k \leq i < j)\Big)\Big]$.

$\mathbf{x}[\mathbf{i}]$ *is to the left of* $\mathbf{x}[\mathbf{j}]$ $\stackrel{\text{df}}{\iff}$

$(\exists k_1 \leq i < k_2 < j \leq k_3)\Big[\mathbf{x}[\mathbf{k_2}]$ is a binary connective and $\mathbf{x}[\mathbf{k_1},\mathbf{k_2}-\mathbf{1}]$ and $\mathbf{x}[\mathbf{k_2}+\mathbf{1},\mathbf{k_3}]$ code formulas and the number of logical symbols other than parentheses in $\mathbf{x}[\mathbf{k_1},\mathbf{k_2}-\mathbf{1}]$ is not less than the number in $\mathbf{x}[\mathbf{k_2}+\mathbf{1},\mathbf{k_3}]\Big]$, or

$(\exists k_1 \leq j < k_2 < i \leq k_3)\Big[\mathbf{x}[\mathbf{k_2}]$ is a binary connective and $\mathbf{x}[\mathbf{k_1},\mathbf{k_2}-\mathbf{1}]$ and $\mathbf{x}[\mathbf{k_2}+\mathbf{1},\mathbf{k_3}]$ code formulas and the number of logical symbols other than parentheses in $\mathbf{x}[\mathbf{k_2}+\mathbf{1},\mathbf{k_3}]$ is greater than the number in $\mathbf{x}[\mathbf{k_1},\mathbf{k_2}-\mathbf{1}]\Big]$.

$\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{j}]$ $\stackrel{\text{df}}{\iff}$ $\mathbf{x}[\mathbf{i}]$ is in the scope of or to the left of $\mathbf{x}[\mathbf{j}]$

The PLOF formula which is equivalent to the infix formula coded by $\mathbf{x}$ can now be defined by

$\mathbf{x}[\mathbf{j}]$ *is the* k*-th PLOF symbol* $\stackrel{\text{df}}{\iff}$ $\mathbf{x}[\mathbf{j}]$ is not a parenthesis and there are $k-1$ values of i such that $\mathbf{x}[\mathbf{i}]$ is not a parenthesis and is before $\mathbf{x}[\mathbf{j}]$

Of course what this last definition says is that the j-th logical symbol of $\mathbf{x}$ becomes the k-th symbol in the natural PLOF formula equivalent to $\mathbf{x}$. This now immediately gives polynomial size formulas for defining the propositional values which code the PLOF formula equivalent to $\mathbf{x}$.

In addition to having polynomial size formulas describing the transformation of an infix formula into PLOF notation we must also have polynomial size Frege proofs of simple properties of the transformation. This is the content of the next two lemmas.

Lemma 7 *('before' is a strict, total ordering on logical symbols)*

(1) $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{j}]$ *and* $\mathbf{x}[\mathbf{j}]$ *is before* $\mathbf{x}[\mathbf{k}]$ *then* $\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{k}]$*."*

(2) $\mathcal{F} \vdash^* $ *"If* $i \neq j$ *and* $\mathbf{x}[\mathbf{i}]$ *and* $\mathbf{x}[\mathbf{j}]$ *are not parentheses then either* $\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{j}]$ *or* $\mathbf{x}[\mathbf{j}]$ *is before* $\mathbf{x}[\mathbf{i}]$*, but not both."*

Lemma 8 *('before' respects subformulas)*

(1) $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}]$ *and* $\mathbf{x}[\mathbf{j}]$ *are in a subformula* $\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}]$ *of* $\mathbf{x}$ *and* $\mathbf{x}[\mathbf{k}]$ *is not, then* $\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{k}]$ *if and only if* $\mathbf{x}[\mathbf{j}]$ *is before* $\mathbf{x}[\mathbf{k}]$*."*

(2) $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}]$ *and* $\mathbf{x}[\mathbf{j}]$ *are in a subformula* $\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}]$ *of* $\mathbf{x}$ *then* $\mathbf{x}[\mathbf{i}]$ *is before* $\mathbf{x}[\mathbf{j}]$ *in* $\mathbf{x}$ *if and only if* $(\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}])[\mathbf{i} + \mathbf{1} - \mathbf{k_1}]$ *is before* $(\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}])[\mathbf{j} + \mathbf{1} - \mathbf{k_1}]$ *in* $\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}]$*."*

Lemmas 7 and 8 are proved with the use of the Unique Readability Lemma above. This is straightforward but tedious and we omit the proof.

$\mathbf{x}[\mathbf{i}]$ *is a predecessor of* $\mathbf{x}[\mathbf{j}] \overset{\text{df}}{\iff}$ $\mathbf{x}[\mathbf{i}]$ is before $\mathbf{x}[\mathbf{j}]$ and there is no $\mathbf{x}[\mathbf{k}]$ before $\mathbf{x}[\mathbf{j}]$ with $\mathbf{x}[\mathbf{i}]$ before $\mathbf{x}[\mathbf{k}]$

$\mathbf{x}[\mathbf{i}]$ *is a successor of* $\mathbf{x}[\mathbf{j}] \overset{\text{df}}{\iff}$ $\mathbf{x}[\mathbf{j}]$ is a predecessor of $\mathbf{x}[\mathbf{i}]$

Lemma 9 *(Discreteness of the 'before' ordering)*

(1) $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}]$ *is not a parenthesis then either* $\mathbf{x}[\mathbf{i}]$ *has a unique predecessor or* $\mathbf{x}[\mathbf{i}]$ *is before every other logical symbol in* $\mathbf{x}$*."*

(2) $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}]$ *is not a parenthesis then either* $\mathbf{x}[\mathbf{i}]$ *has a unique successor or every other logical symbol in* $\mathbf{x}$ *is before* $\mathbf{x}[\mathbf{i}]$*."*

Proof (Sketch) The $\mathcal{F}$-proof of the statements in (1), (2) and (3) proceeds by proving the statements for all subformulas $\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}]$ of $\mathbf{x}$. First the statements are proved for atomic subformulas; this is trivial as there is only one logical symbol in an atomic formula. Then the statements are proved for all $\mathbf{x}[\mathbf{k_1}, \mathbf{k_2}]$ with $k_2 - k_1$ equal to $1, 2, 3, \cdots, n-1$ successively. The proof for each subformula uses the earlier obtained result for its subformulas together with Lemmas 7 and 8. Note how this resembles a proof by induction; however, there is one very important distinction: $\mathcal{F}$ doesn't have induction axioms, instead it proves the statements for all subformulas exhaustively. We call this kind of argument a "brute force induction" on $k_2 - k_1$. $\square$

Lemma 10 $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}$ *codes an infix formula then the string* $\mathbf{y}$ *such that the* k*-th symbol of* $\mathbf{y}$ *is the* k*-th PLOF symbol of* $\mathbf{x}$ *is a PLOF formula. Furthermore, every subformula* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *corresponds to a PLOF subformula* $\mathbf{y}[\mathbf{i}',\mathbf{j}']$ *with* $\mathbf{y}[\mathbf{i}',\mathbf{j}']$ *the natural PLOF translation of* $\mathbf{x}[\mathbf{i},\mathbf{j}]$*."*

Lemma 10 is proved by a brute force "induction" on the length of $\mathbf{x}$ in much the same manner as Lemma 9.

Lemma 11 *If* $\mathcal{F} \vdash^* Con_{\text{PLOF-}\mathcal{F}_+}(\mathbf{x})$ *then* $\mathcal{F} \vdash^* Con_{\mathcal{F}_+}(\mathbf{x})$*.*

Proof The $\mathcal{F}$-proof of $Con_{\mathcal{F}_+}(\mathbf{x})$ proceeds as follows: Suppose $\mathbf{x}$ codes an $\mathcal{F}_+$-proof of $p_0 \wedge \neg p_0$. Convert every formula appearing in $\mathbf{x}$ to PLOF notation. The result is a PLOF-$\mathcal{F}_+$ proof of $\neg p_0 p_0 \wedge$ since $\mathcal{F}_+$-axioms translate w.l.o.g. to PLOF-$\mathcal{F}_+$ axioms and each infix modus ponens inference becomes one of the two forms of PLOF modus ponens. This PLOF-$\mathcal{F}_+$ proof has length less than the proof coded by $\mathbf{x}$ and is coded by a sequence φ of propositional formulas (with variables $\mathbf{x}$). But if $\mathcal{F} \vdash^* Con_{\text{PLOF-}\mathcal{F}_+}(\mathbf{x})$ there is a polynomial size proof of $Con_{\text{PLOF-}F_+}(\varphi)$. This is a contradiction so the assumption that $\mathbf{x}$ codes an $\mathcal{F}_+$-proof of a contradiction is false. $\square$

Hence to prove Main Theorem 1 it will suffice to show $\mathcal{F}$ has polynomial size proofs of $Con_{\text{PLOF-}\mathcal{F}_+}$.

3 Truth Definition for Propositional Formulas

Assume that we have formulas coded by the values of propositional variables $x_1, \cdots, x_N$ in the sense of section 2. Furthermore a formula coded by $\mathbf{x}$ is presumed to involve only variables from $p_0, \ldots, p_m$. A truth definition for such propositional formulas is itself a propositional formula; its free variables are the variables $\vec{x}$ and $\vec{p}$ and its value is equal to true or false according to the value that the formula coded by $\mathbf{x}$ has when the variables p_i are given their assigned values. Note that the variables p_i are being used in two ways: first their names occur in the formula coded by $\mathbf{x}$ and second their values are used in the truth definition. Obviously, for any fixed values of N and m such truth definitions exist; what is also true is that the truth

definitions can be polynomial size in N (assuming m is bounded by some polynomial of N or even $m = N$).

This section describes how the truth of a PLOF formula in the language of $\mathcal{F}_+$ can be expressed by a polynomial size $\mathcal{F}$-formula; furthermore this truth definition will be shown to be intensional; which means that $\mathcal{F}$ can prove with polynomial size proofs that the truth definition properly respects the logical connectives. Although we shall not prove it here, similar results hold for formulas in larger languages with k-ary connectives where $k > 2$ — the techniques are the same as used in the proof that parenthesis languages are in alternating logarithmic time (Buss [1]). The essential technique for the truth definitions was first used in Buss [1] and in Buss-Cook-Gupta-Ramachandran [3] (hereafter: BCGR) where it is shown that evaluation of propositional (Boolean) sentences can be done in alternating logarithmic time.

The common ground between our $\mathcal{F}$-intensional truth definition for propositional formulas and the alternating logarithmic time algorithm for recognizing true propositional sentences is that both require the existence of polynomial size propositional formulas which define the truth of propositional sentences. The alternating logarithmic time algorithm further requires that the polynomial size propositional formulas be *uniform* in the sense of uniform circuits. (Technically speaking, we need U_{E^*}-uniformity. This means that the extended connection language of the formulas must be in alternating log time; or in other words, viewing the formula as a tree, there is an alternating log time algorithm which, given a path from the root to another node in the tree, determines the logical connective at that node.) This uniformity is not required for the work in this paper; however, it is required that the truth definition be $\mathcal{F}$-intensional.

Our construction below of polynomial size propositional formulas defining the truth of prositional formulas is slightly simpler than the prior constructions of Buss and BCGR. The polynomial size formulas we construct are, in fact, U_{E^*}-uniform and this gives a (slightly) new proof that the Boolean formula value problem is in alternating log time. Not only is our proof slightly simpler, but the size of the propositional formulas defining truth of propositional formulas is somewhat smaller and the corresponding alternating log time algorithm for recognizing true propositional sentences is somewhat more efficient. This simpler alternating log time algorithm for the Boolean formula value problem makes the details of the definitions and proofs of this section substantially easier.

To ensure $\mathcal{F}$-intensionality, we use a technique from section 6 of BCGR [3] to restrict attention to $\leq$1-scarred formulas. The use of $\leq$1-scarred formulas was necessary in BCGR's construction of logarithmic depth arithmetic circuits for evaluating arithmetic formulas over rings or fields; for us, using $\leq$1-scarred subformulas is an important tool for showing that our truth definition for propositional formulas is $\mathcal{F}$-intensional.

We shall give the truth definition only for PLOF formulas; however, the translation of infix formulas to PLOF formulas could be used to extend it to a truth definition of infix formulas.

Definition A *1-scarred* postfix formula is a string w of symbols such that p_0w is a postfix formula. A *1-scarred PLOF* formula is a 1-scarred postfix formula such that $p_0\neg\neg\cdots\neg w$ is a PLOF formula for sufficiently large number of $\neg$'s.

A $\leq$1-scarred postfix (PLOF) formula is a string which is either a postfix (PLOF) formula or a 1-scarred postfix (PLOF) formula.

The idea is that 1-scarred (PLOF) formula is a (PLOF) formula with one, leftmost subformula removed; the point at which the leftmost subformula is detached is called a "scar". For the rest of this section, all formulas (scarred or otherwise) are taken to be PLOF formulas.

The truth definition of formulas will define the truth of a formula in terms of truth values for $\leq$1-scarred subformulas. The truth value of a $\leq$1-scarred subformula $\mathbf{w}$ will be a pair (t_1, t_2) of truth values. If $\mathbf{w}$ is a formula (with no scars) then $t_1 = t_2$ is the truth value of $\mathbf{w}$; otherwise, if $\mathbf{w}$ has one scar then t_1 is the truth value of $\top\mathbf{w}$ and t_2 is the truth value of $\bot\mathbf{w}$. The composition symbol $\circ$ is used to combine truth values of $\leq$1-scarred formulas; namely, if $\mathbf{v}$ is $\leq$1-scarred and has truth value (s_1, s_2) and if $\mathbf{w}$ is 1-scarred and has truth value (t_1, t_2) then their concatenation $\mathbf{vw}$ is $\leq$1-scarred and has truth value $(r_1, r_2) = (s_1, s_2) \circ (t_1, t_2)$ where $\circ$ is defined so that

$$r_i = \begin{cases} t_1 & \text{if } s_i = \top \\ t_2 & \text{if } s_i = \bot \end{cases}$$

We now embark upon the definition of the polynomial size, propositional truth definition of $\leq$1-scarred subformulas. Keeping things polynomial size requires a rather complicated way of splitting a $\leq$1-scarred subformula into $\leq$1-scarred subformulas; the worst complication is that we must refer to the

subformulas in an indirect way. We will begin by defining "breakpoints" of a $\leq$1-scarred formula inside an interval. We will then use breakpoints inside a $\leq$1-scarred formula is split it into up to four $\leq$1-scarred subformulas. The truth definition will define the truth of a $\leq$1-scarred formula in terms of the truth values of the $\leq$1-scarred subformulas obtained this way.

Definition Let $\mathbf{x}$ code a postfix formula. Then $\mathbf{x_j}$ is the unique subformula of $\mathbf{x}$ of the form $\mathbf{x[i,j]}$. If $\mathbf{x}$ does not code a postfix formula (for example, $\mathbf{x}$ might code a proof) then $\mathbf{x_j}$ is the unique subformula of the form $\mathbf{x[i,j]}$ if such a formula exists and is undefined otherwise.

Definition Let $\mathbf{x}$ code a string of symbols (say a PLOF-$\mathcal{F}_+$-proof) and $\mathbf{x[i,j]}$ be a $\leq$1-scarred formula. Suppose $\ell < r$, $\ell < j$ and $i \leq r$. We say k *is the breakpoint of* $\mathbf{x[i,j]}$ *1-selected by* $(\ell, r]$ provided k is the largest value $\leq \min\{r, j\}$ such that $\mathbf{x_k}$ contains one of the symbols $\mathbf{x}[\ell + \mathbf{1}]$ or $\mathbf{x[i]}$. In other words, $\mathbf{x_k}$ must be of the form $\mathbf{x[m,k]}$ with

$$m \leq \max\{i, \ell + 1\} \leq k \leq \min\{r, j\}$$

and k must be maximum so that this holds. Basically, $\mathbf{x[k]}$ is to be the rightmost connective up to $\mathbf{x}[\mathbf{min\{r,j\}}]$ which has $\mathbf{x}[\mathbf{max\{i}, \ell + \mathbf{1\}}]$ in its scope (or is equal to $\mathbf{x}[\mathbf{max\{i}, \ell + \mathbf{1\}}]$).

In the degenerate case $j \leq \ell$ (or $r < i$, respectively) the breakpoint of $\mathbf{x[i,j]}$ 1-selected by $(\ell, r]$ is defined to be j (respectively, $i - 1$).

Definition Let $\mathbf{x}$ code a postfix formula. We say that i is an *ancestor* of j, $j \trianglelefteq i$, if and only if the symbol $\mathbf{x[j]}$ is in the formula $\mathbf{x_i}$, or in other words, if and only if $\mathbf{x[j]}$ is in the scope of $\mathbf{x[i]}$ or $i = j$. The *least common ancestor* (l.c.a.) of i and j is the least value k such that $\mathbf{x_k}$ contains both $\mathbf{x[i]}$ and $\mathbf{x[j]}$, i.e., the least k such that $i \trianglelefteq k$ and $j \trianglelefteq k$.

Definition Let Δ_u and ϵ_u be integers defined inductively by:

$$\begin{aligned} \Delta_0 &= 2 \\ \epsilon_u &= \lfloor \tfrac{1}{2}\Delta_u \rfloor \\ \Delta_{u+1} &= \Delta_u + \epsilon_u. \end{aligned}$$

Intuitively, one should think of Δ_u as being approximately equal to $(3/2)^u$ and, indeed, it is easy to prove that $(3/2)^{u+2} > \Delta_u > (3/2)^{u+1}$. Also, it trivial that $\Delta_{u+1} - 2\epsilon_u \geq \epsilon_u$.

We shall only need Δ_u such that $\Delta_u = O(N)$ where $N = |x|_\Sigma$; that is to say, we shall only need Δ_u for $u = O(\log N)$. Although it is not needed for our $\mathcal{F}$-intensional definition of truth of propositional formulas, it is also easy to see that for $u = O(\log N)$, Δ_u can be computed in alternating log time, i.e., in alternating time $O(\log N)$. To prove this, construct circuits of depth $O(u)$ to compute Δ_u: the circuits use carry-save-addition and just use the definitions above to compute Δ_{u+1} from Δ_u in constant depth (as usual in carry-save-addition, Δ_u is represented by a pair of integers whose sum is equal to Δ_u). These circuits are clearly U_{E^*}-uniform.

Definition Let $\mathbf{x}$ code a string of symbols and $\mathbf{x}[\mathbf{i},\mathbf{j}]$ be a formula. Let $n-m$ be equal to Δ_{u+1} for some $u \geq 0$.[4] The *breakpoints of* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *generated by* $(m,n]$ include:

(1) The value a_1 which is the breakpoint of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ 1-selected by $(m, m+\epsilon_u]$.

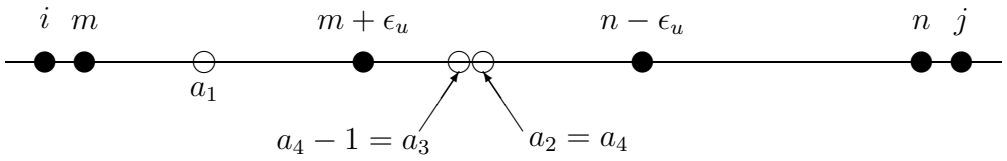
(2) The value a_2 which is the breakpoint of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ 1-selected by $(m+\epsilon_u, n-\epsilon_u]$.

(3) The least common ancestor a_4 of a_1 and a_2, unless $a_1 = i-1$ in which case, $a_4 = a_2$.

(4) The value $a_3 = a_4 - 1$.

Figures 1 and 2 show some examples of how breakpoints might be picked. These are merely representative examples; for instance, $i < m$ and $n < j$ will not always be true. Figure 1 shows an example where $a_1 \trianglelefteq a_2$ and hence $a_4 = a_2$; in this case we could actually dispense with the breakpoints a_1 and a_3 but we shall keep them to avoid having two cases in all the definitions below. Figure 2 shows the more complicated case where $a_4 \neq a_2$. Not shown is the case where $a_3 = a_2 = n - \epsilon_u$.

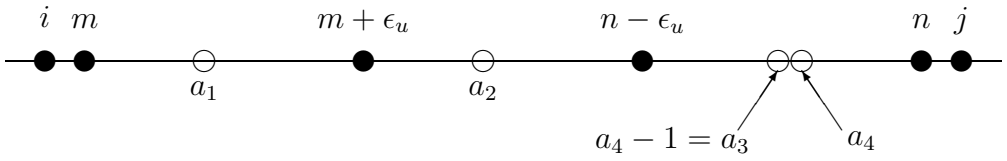
[4]It is possible to modify this definition so as to remove the restriction that $n - m = \Delta_{u+1}$. Basically one would just replace ϵ_u in this definition and in the rest of the paper by $\lfloor \frac{(n-m)}{3} \rfloor$. This would still preserve the alternating log time uniformity of our propositional formulas. The main reason we use the Δ_{u+1}'s and ϵ_u's is to avoid writing too many fractions.



$$\max\{m+1,i\} \trianglelefteq a_1 \trianglelefteq a_2 = a_4 \trianglelefteq j$$

$$m - n = \Delta_{u+1}$$

Figure 1 - Definition of break points; example of $a_1 \trianglelefteq a_2$



$$\max\{m+1,i\} \trianglelefteq a_1 \not\trianglelefteq a_1+1 \trianglelefteq a_2 \trianglelefteq a_4 \trianglelefteq j$$

$$a_4 = l.c.a\{a_1,a_2\} \qquad\qquad m - n = \Delta_{u+1}$$

Figure 1 - Definition of break points; example of $a_1 \not\trianglelefteq a_2$

Before proceeding further with the formal definitions let's examine the motivations for the definitions of breakpoints. Suppose $\mathbf{x}$ codes a proof and $\mathbf{x}[\mathbf{i},\mathbf{j}]$ is a $\leq$1-scarred formula in the proof with $m < i \leq j \leq n$. We want to find the truth value of $\mathbf{x}[\mathbf{i},\mathbf{j}]$; recall this is a pair of truth values.[5] What we want to do is find the up to four breakpoints a_1, a_2, a_3, a_4 of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ generated by $(m,n]$ and use them to split $\mathbf{x}[\mathbf{i},\mathbf{j}]$ into up to four $\leq$1-scarred subformulas. Suppose for the sake of illustration that $a_2 < a_3$; then there are four intervals delineated by the breakpoints; namely, $\mathbf{x}[\mathbf{m}+\mathbf{1},\mathbf{a_1}]$, $\mathbf{x}[\mathbf{a_1}+\mathbf{1},\mathbf{a_2}]$, $\mathbf{x}[\mathbf{a_2}+\mathbf{1},\mathbf{a_3}]$, and $\mathbf{x}[\mathbf{a_4}+\mathbf{1},\mathbf{n}]$. These four intervals completely cover $\mathbf{x}[\mathbf{m}+\mathbf{1},\mathbf{n}]$ with the exception of the binary operator $\mathbf{x}[\mathbf{a_4}]$. Furthermore, we will prove below that these four intervals, when intersected with $\mathbf{x}[\mathbf{i},\mathbf{j}]$ yield (up to) four $\leq$1-scarred subformulas of $\mathbf{x}[\mathbf{i},\mathbf{j}]$. Now, the idea, of course, is to define the truth value of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ in terms of the truth values of these four $\leq$1-scarred subformulas. This is readily done; however, at this point an additional complication arises. The complication is that a straightforward definition of the truth of a formula in terms of the truth of four $\leq$1-scarred subformulas would give a superpolynomial size truth definition. The solution to this difficulty depends, in part, on the fact that we are defining the truth of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *inside the interval* $(m,n]$; in fact this is the sole purpose of mentioning $(m,n]$ at all. Likewise the four $\leq$1-scarred subformulas of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ will be evaluated inside intervals, which will be approximately two-thirds as large as the interval $(m,n]$. Specifically, the truth of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ inside interval $(m,n]$ of $\mathbf{x}$ will be evaluated by the following:

(1) Evaluating $\mathbf{x}[\mathbf{i},\mathbf{a_1}]$ and $\mathbf{x}[\mathbf{a_1}+\mathbf{1},\mathbf{a_2}]$ inside the interval $(m, n-\epsilon_u]$ of $\mathbf{x}$,

(2) Evaluating $\mathbf{x}[\mathbf{a_2}+\mathbf{1},\mathbf{a_3}]$ and $\mathbf{x}[\mathbf{a_4}+\mathbf{1},\mathbf{j}]$ inside the interval $(m+\epsilon_u, n]$ of $\mathbf{x}$,

(3) Combining the values obtained in steps (1) and (2) using composition and the binary connective $\mathbf{x}[\mathbf{a_4}]$ to obtain a truth value of $\mathbf{x}[\mathbf{i},\mathbf{j}]$.

Let's generalize and formalize the above example. Let $\mathbf{x}[\mathbf{i},\mathbf{j}]$ be a $\leq$1-scarred subformula of a formula in $\mathbf{x}$; suppose $n-m = \Delta_{u+1}$ and let $a_1, \ldots, a_4$ be the breakpoints generated by the interval $(m,n]$. Define the *subformulas of* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *inside* $(m,n]$ by:

[5]More specifically, the pair of truth values consists of two propositional formulas which have as free variables the variables $\vec{x}$ and the variables named in the formula coded by $\mathbf{x}$.

$$
\begin{aligned}
SubFm_1(\mathbf{x}, [i,j], (m,n], 1) &= [i, a_1] \\
SubFm_1(\mathbf{x}, [i,j], (m,n], 2) &= [a_1 + 1, a_2] \\
SubFm_1(\mathbf{x}, [i,j], (m,n], 3) &= [a_2 + 1, a_3] \\
SubFm_1(\mathbf{x}, [i,j], (m,n], 4) &= [a_4 + 1, j]
\end{aligned}
$$

If any interval listed above has beginning to the left of its end then that interval is to be *undefined*.[6]

Formally speaking, $SubFm_1$ has arguments $\mathbf{x}$ which codes a string of symbols, a closed interval $[i,j]$, a left open interval $(m,n]$ and an integer in $\{1,\ldots 4\}$ and produces a closed interval as a value. Here we are intending that closed intervals and left open intervals are yet another "sort" and are coded by two integers giving the endpoints (and these two integers are coded by propositional variables in either binary or unary notation). By letting k range over $\{1,2,3,4\}$, $SubFm_1$ gives the intervals of $\mathbf{x}$ containing the *subformulas of* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *inside* $(m,n]$.

Also define the distinguished binary operator by:

$$
BinOp(\mathbf{x}, [i,j], (m,n]) = \begin{cases} \mathbf{x}[\mathbf{a_4}] & \text{if } a_4 \neq a_2 \\ \text{undefined} & \text{otherwise} \end{cases}
$$

Note that $BinOp(\cdots)$ actually is a binary operator if it is defined since it is the least common ancestor of $\mathbf{x}[\mathbf{a_1}]$ and $\mathbf{x}[\mathbf{a_2}]$.

It is easy to see that $SubFm_1(\cdots)$ and $BinOp(\cdots)$ are defined by polynomial size formulas (polynomial in the length N of $\mathbf{x}$) by using the methods of section 2 for parsing postfix formulas. The free variables appearing in $SubFm_1$ and $BinOp$ are the propositional variables $\mathbf{x}$ plus variables coding the integers i,j,m,n,k. (Remark: there would be no essential change if, instead of having the integers i,j,m,n,k as arguments coded by propositional variables, we wrote the integers as subscripts and thought of these values being 'hardwired' into the formulas. The reason this makes no essential change is that there are only polynomially many values these integers can assume.)

[6]There are several reasons why one of the $SubFm_1$ might be undefined. First, some of the breakpoints may coincide by being equal to $i-1$ or equal to j. Even if the breakpoints lie inside $\mathbf{x}[\mathbf{i},\mathbf{j}]$ there are several cases where a $SubFm_1$ might be undefined: specifically, if $a_1 \trianglelefteq a_2$ then $a_4 = a_2$ and $a_3 = a_2 - 1$ (illustrated in Figure 1); another possibility is $a_3 = a_2 = n - \epsilon_u$. We shall later handle undefined subformulas with the convention that they have the identity function $(\top, \bot)$ as truth value.

Definition Let $\mathbf{x}[\mathbf{a}, \mathbf{b}]$ be a substring of $\mathbf{x}$. We say that $\mathbf{x}[\mathbf{i}]$ is a *scar* of the interval $[a, b]$ if and only if $i < a$ and there is a connective $\mathbf{x}[\mathbf{k}]$ with $a \leq k \leq b$ such that $\mathbf{x_i}$ is one of the operands of $\mathbf{x}[\mathbf{k}]$.

Note that if $i \neq a - 1$ or $k \neq a$, then $\mathbf{x}[\mathbf{k}]$ must be a binary connective and $\mathbf{x_i}$ its first operand.

Lemma 12 $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ *is a* ≤ 1*-scarred formula, if* $n - m = \Delta_{u+1}$*, and if* $a_1, \ldots, a_4$ *are the breakpoints from the definition of* $SubFm_1(\mathbf{x}, [i, j], (m, n], p)$ *then*

(a) *If* $a_1 \neq i - 1$ *(equivalently,* $i \leq m + \epsilon_u$*), then* $a_1 + 1 \trianglelefteq a_2$*.*

(b) *For every* a *such that* $\max\{m + 1, i\} \leq a \leq a_2$*, either* $a \trianglelefteq a_1$ *or* $a \trianglelefteq a_2$*. Similarly, for every* a *such that* $\max\{m + 1, i\} \leq a \leq a_3$*, either* $a \trianglelefteq a_1$ *or* $a \trianglelefteq a_2$ *or* $a \trianglelefteq a_3$*.*

(c) *For* $p = 1, 2, 3, 4$*,* $SubFm_1(\mathbf{x}, [i, j], (m, n], p)$*, if defined, does not have more than one scar* $\mathbf{x}[\mathbf{k}]$ *with* $k \geq \max\{m + 1, i\}$*.*

Proof We shall argue informally to prove the quoted material; however, it will be clear that our arguments can be formalized as polynomial size $\mathcal{F}$-proofs using the fact that Frege systems can prove elementary syntactic facts regarding PLOF formulas. To prove (a), suppose for sake of a contradiction, that $i \leq m + \epsilon_u$ and $a_1 + 1 \not\trianglelefteq a_2$. Then $\mathbf{x_{a_2}}$ is equal to $\mathbf{x}[\mathbf{b}, \mathbf{a_2}]$ with $a_1 + 1 < b$. By the definition of a_2, $m + \epsilon_u + 1 \trianglelefteq a_2$, so $b \leq m + \epsilon_u + 1$. Now, $\mathbf{x_{b-1}}$ must be equal to $\mathbf{x}[\mathbf{c}, \mathbf{b - 1}]$ with $a_1 < c$; since otherwise $a_1 \trianglelefteq b - 1 \leq m + \epsilon_u$ would violate the maximality of a_1 as defined by 1-selection. Let d be the least common ancestor of $b - 1$ and a_2; obviously, $d > a_2$. Also, it must be that $d > n - \epsilon_u$ since $a_2 \trianglelefteq d$ and $d \leq n - \epsilon_u$ would violate the maximality of a_2 as defined by 1-selection. Clearly $\mathbf{x}[\mathbf{d}]$ is a binary connective with first (left) operand $\mathbf{x}[\mathbf{c}, \mathbf{b - 1}]$ and second (right) operand $\mathbf{x}[\mathbf{b}, \mathbf{d - 1}]$. Now,

$$m < a_1 < c < b \leq m + \epsilon_u + 1 \leq a_2 \leq n - \epsilon_u < d.$$

Hence $\mathbf{x}[\mathbf{d}]$'s first operand has lenth $b - c < \epsilon_u$ (note that $m + 2 \leq c$) and $\mathbf{x}[\mathbf{d}]$'s second operand has length $d - b > (n - \epsilon_u) - (m + \epsilon_u) = \Delta_{u+1} - 2\epsilon_u \geq \epsilon_u$. In other words, $\mathbf{x}[\mathbf{d}]$'s first operand is shorter than its second operand which contradicts the fact that $\mathbf{x}$ codes a PLOF formula.

Part (b) is an easy consequence of $a_1 + 1 \trianglelefteq a_2$ and of $\max\{m+1, i\} \trianglelefteq a_1$ and of the fact that if $a_2 \neq a_4$, then $a_2 \trianglelefteq a_3$. (We omit the proof of the even easier case where $i > m + \epsilon_u$.)

To prove part (c), first note that $SubFm_1(\mathbf{x}, [i,j], (m,n], 1)$ is equal to $[i, a_1]$ if it is defined, and trivially has no scar $k \geq \max\{m+1, i\}$. This also implies that if $a_1 \trianglelefteq a_2$ then $SubFm_1(\mathbf{x}, [i,j], (m,n], 2)$ has a scar at $k = a_1$ but has no other scar $k \geq \max\{m+1, i\}$. On the other hand, if $a_1 \not\trianglelefteq a_2$ then by part (a), $\mathbf{x}[\mathbf{a_2}] = \mathbf{x}[\mathbf{a_1} + \mathbf{1}, \mathbf{a_2}]$ is an (unscarred) formula. $SubFm_1(\mathbf{x}, [i,j], (m,n], 3)$ is equal to $[a_2 + 1, a_3]$ if defined. This clearly has a scar at a_2: we claim this is the only one. To see this, note that by part (b) the only other candidate for a scar is a_1; but $\mathbf{x}_{\mathbf{a_1}}$ is the first operand to $\mathbf{x}[\mathbf{a_4}]$ so $a_1 \not\trianglelefteq a_3$ and hence a_1 is not a scar of $[a_2 + 1, a_3]$. Finally, $SubFm_1(\mathbf{x}, [i,j], (m,n], 4)$ is equal to $[a_4 + 1, j]$ if defined. This has a scar at a_4 and we claim this is the only one. To see this, again note that by (b), the only possible other scars are a_1, a_2 and a_3. However, $a_1 \trianglelefteq a_4$, $a_2 \trianglelefteq a_4$ and $a_3 \trianglelefteq a_4$, so they are not scars of $[a_4 + 1, j]$. $\square$

It is interesting to note that the proof of Lemma 12(a) is the only place we ever use the fact that $\mathbf{x}$ codes a PLOF formula instead of an ordinary postfix formula.

$SubFm_1$ picks out up to four $\leq$1-scarred subformulas of a subformula; we now need to iterate this process and pick out $\leq$1-scarred subformulas of these $\leq$1-scarred subformulas, and so on. More specifically, if $p_1, \ldots, p_k \in \{1,2,3,4\}$, we use $SubFm_1$ to pick out the p_1-th subformula of $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ then use $SubFm_1$ again to obtain the p_2-th subformula of the result, and so on for k steps. For this, suppose $n - m = \Delta_{u+1}$, then define $Int_0((m,n])$ to be equal to $(m,n]$ and for all $k \leq u+1$, if $Int_{k-1}((m,n], p_1, \ldots, p_{k-1})$ is equal to $(m', n']$ then

$$Int_k((m,n], p_1, \ldots, p_k) = \begin{cases} (m', n' - \epsilon_{u+1-k}] & \text{if } p_k = 1, 2 \\ (m' + \epsilon_{u+1-k}, n'] & \text{if } p_k = 3, 4. \end{cases}$$

Int_k, of course, defines an object of the left open interval sort: it is to be used as the interval inside which breakpoints are generated. It is easy to see by induction on k that $Int_k((m,n], p_1, \ldots, p_k)$ is a interval $(m'', n'']$ with $n'' - m'' = \Delta_{u+1-k}$. Thus $Int_k(\cdots)$ is of length approximately two-thirds the length of $Int_{k-1}(\cdots)$.

What we wish to accomplish is to define $SubFm_{k+1}$ so that

$$\begin{aligned} &SubFm_{k+1}(\mathbf{x}, [i,j], (m,n], p_1, \ldots, p_{k+1}) = \\ &\quad = SubFm_1(\mathbf{x}, SubFm_k(\mathbf{x}, [i,j], (m,n], p_1, \ldots, p_k), \qquad (*) \\ &\qquad\qquad Int_k((m,n], p_1, \ldots, p_k), p_{k+1}) \end{aligned}$$

The idea is that this specifies a $\leq$1-scarred subformula of $\mathbf{x}$ which is to be evaluated as part of the process of assigning a truth value to $\mathbf{x}[\mathbf{i}, \mathbf{j}]$. Unfortunately, using (*) as the definition of the formulas $SubFm_{k+1}$ makes the formulas have superpolynomial size. To see this note that $SubFm_1$ has size $N^{O(1)}$ (i.e., is polynomial size in the length N of $\mathbf{x}$) and by inspection, it uses its second argument (the closed interval) polynomially many times. If (*) were adopted as a definition, $SubFm_{k+1}$ would consist of $SubFm_1$ with $SubFm_k$ as its second argument and then, by induction on k, $SubFm_k$ would have size $N^{O(k)}$; but k will range up to $\log_2(N)$ and this would make the truth definition be a formula of superpolynomial size $N^{O(\log_2 N)}$.[7]

Fortunately, we can give a polynomial size definition of $SubFm_{k+1}$ by calculating the breakpoints of $SubFm_k(\cdots)$ inside the interval $Int_k((m,n], \vec{p})$ in a manner that is independent of $SubFm_k(\cdots)$. We begin by observing that Int_k can be defined by polynomial size formulas. This is because, if $n - m = \Delta_{u+1}$, then $Int_k((m,n], p_1, \ldots, p_k)$ will be equal to an interval $(m', n']$ where

$$\begin{aligned} m' &= m + \sum_{j=1}^{k} \left\lfloor \tfrac{1}{2}(p_j - 1) \right\rfloor \cdot \epsilon_{u+1-j}, \\ n' &= m' + \Delta_{u+1-k}. \end{aligned}$$

The point of the overly mysterious $\lfloor \frac{1}{2}(p_j - 1) \rfloor$ is that it will equal 0 or 1 depending on whether the lower part ($p_j = 1, 2$) or upper part ($p_j = 3, 4$) is selected. Since vector summation is definable with polynomial size formulas, it is immediate that Int_k is definable with polynomial size formulas.[8] We

[7] This was the crux of the difficulty involved in giving a polynomial size truth definition for propositional formulas or, more-or-less equivalently, an alternating logarithmic time algorithm for recognizing true propositional sentences. As we have already remarked, the use of the interval $(m, n]$ and its subintervals given by Int_k is what allows us to get the polynomial size truth definition and the alternating logarithmic time algorithm.

[8] The values Δ_u and ϵ_u may either be 'hardwired' into the formula or be computed by U_{E^*}-uniform polynomial size formulas as discussed earlier.

can now give a definition of $SubFm_k$ which does not use the definition of $SubFm_1$ iteratively:

Definition (of $SubFm_k$). Fix $p_1, \ldots, p_k$. For $\ell \leq k$, let $a_1^\ell, \ldots, a_4^\ell$ be the (up to) four breakpoints of $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ generated by the interval $Int_{\ell-1}((m,n], p_1, \ldots, p_{\ell-1})$. Also, let $a_0^\ell = i - 1$ and $a_5^\ell = j$ for all ℓ. Define i_ℓ by

$$i_\ell \;=\; \begin{cases} p_\ell - 1 & \text{if } 1 \leq p_\ell \leq 3 \\ 4 & \text{if } p_\ell = 4 \end{cases}$$

Comparing this definition of i_ℓ to the definition of the subformulas of $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ inside $(m, n]$, it is clear that $SubFm_1(\mathbf{x}, [i, j], Int_{\ell-1}((m, n], \vec{p}), p_\ell)$ is equal to $[a_{i_\ell}^\ell + 1, a_{1+i_\ell}^\ell]$. Further define

$$\begin{aligned} c_k &= \max\{a_{i_\ell}^\ell : 1 \leq \ell \leq k\} \\ d_k &= \min\{a_{1+i_\ell}^\ell : 1 \leq \ell \leq k\}. \end{aligned}$$

Now, $SubFm_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is defined to be equal to $[c_k + 1, d_k]$.

Thus $SubFm_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is the intersection of the interval $SubFm_{k-1}(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_{k-1})$ and the interval $[a_{i_\ell}^\ell + 1, b_{1+i_\ell}^\ell]$. That is to say, $SubFm_k(\cdots, p_{k-1}, p_k)$ is the substring of $SubFm_{k-1}(\cdots, p_{k-1})$ delimitated by the appropriate breakpoints generated by $Int_{k-1}(\cdots, p_{k-1})$.

We say that $SubFm_k(\cdots)$ is *defined* if, according to the above definition, $SubFm_k(\cdots)$ is equal to $[a, b]$ with $a \leq b$; if, however, $a > b$ then we say it is *undefined*.

It is straightforward to check that this definition makes $SubFm_k$ a (vector of) polynomial size formula(s). This is because (1) the binary representation of a_i^ℓ can be defined by polynomial size formulas since the a_i^ℓ's are defined in terms of breakpoints generated inside $Int_{\ell-1}(\cdots)$ which we already established to have polynomial size formulas, and (2) the maximum or minimum of polynomially many integers is easily describable via polynomial size formulas.

We have completed the definition of how to generate breakpoints of $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ inside $(m, n]$. Next we need to show that $SubFm_k(\cdots)$ actually does give $\leq$1-scarred subformulas and that these subformulas fully specify the truth value of $\mathbf{x}[\mathbf{i}, \mathbf{j}]$. The next lemma gives some requisite technical properties of $SubFm_k$'s.

Lemma 13 $\mathcal{F} \vdash^*$ *"If* $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ *is a* $\leq$*1-scarred subformula, if* $n - m$ *is equal to* Δ_{u+1}*, if* $k \geq 0$*, if* $p_1, \ldots, p_k \in \{1, \ldots, 4\}$*, and if* A *denotes the interval*

$$SubFm_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$$

then:

(a) A *is properly contained in* $Int_k((m, n], p_1, \ldots, p_k)$*.*

(b) *Each symbol in* A *is in exactly one of the intervals*

$$SubFm_{k+1}(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_{k+1})$$

or is the binary operator

$$BinOp(\mathbf{x}, A, Int_k(\mathbf{x}, (m, n], p_1, \ldots, p_k)).$$

(c) *Each* $SubFm_{k+1}(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_{k+1})$ *is a* $\leq$*1-scarred subformula."*

Proof (Sketch). Part (a) is proved by brute force "induction" on k using the fact that in the definition of breakpoints, $m + \epsilon_u < a_2 \leq n - \epsilon_u$. Part (b) follows immediately from the definition of $SubFm$. The proof of (c) is another brute force "induction" on k. Letting $SubFm_0(\mathbf{x}, [i, j], (m, n]) = [i, j]$, the base case with $k = 0$ is immediate. For the induction step, suppose the lemma is already proved for k: From the definition of $SubFm_j$, we know that $SubFm_j(\mathbf{x}, [i, j], (m, n], \vec{p})$ is equal to $[c_j + 1, d_j]$ and that $c_{k+1} = \max\{c_k, a^{k+1}_{i_{k+1}}\}$ and $d_{k+1} = \min\{d_k, a^{k+1}_{1+i_{k+1}}\}$; in other words, $SubFm_{k+1}(\mathbf{x}, [i, j], (m, n], \vec{p}, p_{k+1})$ is equal to the intersection of $[a^{k+1}_{i_{k+1}} + 1, a^{k+1}_{1+i_{k+1}}]$ and $SubFm_k(\mathbf{x}, [i, j], (m, n], \vec{p})$. Now the latter of these is contained inside $Int_k((m, n], \vec{p})$ by part (a) and is a $\leq$1-scarred formula by the induction hypothesis. And the former, by Lemma 12(c), has at most one scar inside $Int_k((m, n], \vec{p})$. So to prove (c) it suffices to show that the intersection of two $\leq$1-scarred subformulas is $\leq$1-scarred.

To prove this last claim, suppose $\mathbf{x}[\mathbf{i_1}, \mathbf{j_1}]$ and $\mathbf{x}[\mathbf{i_2}, \mathbf{j_2}]$ are $\leq$1-scarred formulas and. w.l.o.g., $i_1 \leq i_2$. If $\mathbf{x}[\mathbf{i}_\ell, \mathbf{j}_\ell]$ is in fact 1-scarred, then its scar must be at $\mathbf{x}[\mathbf{i}_\ell - \mathbf{1}]$. Any scar of the intersection $\mathbf{x}[\mathbf{i_2}, \mathbf{min}\{\mathbf{j_1}, \mathbf{j_2}\}]$ must also be a scar of $\mathbf{x}[\mathbf{i_2}, \mathbf{j_2}]$ so the intersection can have at most one scar (which will be at $\mathbf{x}[\mathbf{i_2} - \mathbf{1}]$ if it exists). $\Box$

Lemma 13 contains all the crucial technical prerequisites for our definition of the truth value of a $\leq$1-scarred subformula. Given a $\leq$1-scarred subformula $\mathbf{x}[\mathbf{i},\mathbf{j}]$ and given m, n with $n - m$ equal to Δ_{u+1} and $m < i \leq j \leq m$ we are now ready to define the *truth value of* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *inside* $(m, n]$; recall that the truth value is to be a pair of Boolean truth values. To define truth, we shall define polynomial size formulas $Value_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ which is to be the truth value of the $\leq$1-scarred subformula in the interval $SubFm_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$. Taking $k = 0$ we get the truth value of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ inside $(m, n]$.

Definition Let $n - m = \Delta_{u+1}$ and $m < i \leq j \leq n$ with $\mathbf{x}[\mathbf{i},\mathbf{j}]$ a $\leq$1-scarred formula. The variables k and p_k will range over values $0 \leq k \leq u + 1$ and $1 \leq p_k \leq 4$. $Value_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is defined by:

Case (1): $k = u + 1$. So $Int_k(\mathbf{x}, (m, n], p_1, \ldots, p_k) = (a, a + 2]$ for some a. By Lemma 13(a), $SubFm_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ will either be undefined or be the single logical symbol $\mathbf{x}[\mathbf{a + 1}]$ or $\mathbf{x}[\mathbf{a + 2}]$. If $SubFm_k(\mathbf{x}, [i, j], (m, n], \vec{p})$ is undefined then $Value(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is defined to be $(\top, \bot)$. Otherwise, $SubFm_k(\cdots)$ is consists of a single logical symbol and must be either "$\neg$" or "$\top$" or "$\bot$" or a variable, say "q". In this case, $Value_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is defined to be $(\bot, \top)$ or $(\top, \top)$ or $(\bot, \bot)$ or (q, q), respectively. By (q, q) we mean $(\top, \top)$ if q has truth value *True* and $(\bot, \bot)$ if q has truth value *False*.[9]

Cases (2) and (3): $k \leq u$. Let $Int_k(\mathbf{x}, [i, j], (m, n], \vec{p})$ be equal to $(a, b]$. For $1 \leq p_{k+1} \leq 4$, let $I_{p_{k+1}}$ be $SubFm_{k+1}(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k, p_{k+1})$. By the definition of $SubFm_{k+1}$, the $\leq$1-scarred subformulas $I_{p_{k+1}}$ can be obtained by using the four breakpoints a_1, a_2, a_3, a_4 of $\mathbf{x}[\mathbf{i},\mathbf{j}]$ generated by $(a, b]$ which split $SubFm_k(\mathbf{x}, [i, j], (m, n], \vec{p})$ into four intervals I_1, I_2, I_3, I_4.

Case (2): Suppose $a_2 = a_4$. Then $Value_k(\mathbf{x}, [i, j], (m, n], p_1, \ldots, p_k)$ is defined to be

$$Value_{k+1}(\cdots, 1) \circ Value_{k+1}(\cdots, 2) \circ Value_{k+1}(\cdots, 4)$$

[9]Recall that the truth definition was to involve variables that are named in the formula coded by $\mathbf{x}$; this case is where such variables come in.

where "$\cdots$" stands for "$\mathbf{x}, [i,j], (m,n], p_1, \ldots, p_k$"; so 1, 2, 4 are values for p_{k+1}. Recall that $\circ$ means (reverse) composition.

Case (3): Suppose $a_2 \neq a_4$. Then $Value_k(\mathbf{x}, [i,j], (m,n], p_1, \ldots, p_k)$ is defined to be

$$f_{BinOp}\Big(Value_{k+1}(\cdots, 1), Value_{k+1}(\cdots, 2) \circ Value_{k+1}(\cdots, 3)\Big) \circ \\ \circ Value_{k+1}(\cdots, 4)$$

where if $BinOp(\mathbf{x}, [i,j], Int_k((m,n], p_1, \ldots, p_k)$ is the connective $\circledcirc$, then

$$f_{BinOp}((s_1, s_2), (t,t)) = (s_1 \circledcirc t, s_2 \circledcirc t).$$

The definition of $Value_k$ has a base case and two inductive cases. Note that undefined intervals (e.g., for breakpoints outside of the interval $[i,j]$ or for intervals of length zero) are given the identity function as value — this was explicitly stated in the base case and propogated upwards through the inductive cases. This convention allowed us to avoid having to enumerate all the various ways that $\leq$1-scarred subformulas might or might not be contained in the interval $[i,j]$ and simplified the definition considerably.

The above completes the truth definition for $\leq$1-scarred formulas. Of course, by the translation of infix formulas into PLOF form, this also gives a truth definition for Boolean formulas in the usual infix form.

We have claimed all along that the above definitions were $\mathcal{F}$-intensional; i.e., $\mathcal{F}$ can prove basic facts about how we parse formulas and define truth. Of course we have been careful to make sure that the formulas were polynomial size but there still remains to show the crucial fact that there are polynomial size $\mathcal{F}$-proofs of the fact that (1) our truth definition for PLOF formulas respects the meanings of the propositional connectives and (2) the value $Value_0(\mathbf{x}, [i,j], (m,n])$ of a formula $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ is independent of m and n provided $n - m$ is equal to Δ_{u+1} and $m < i \leq j \leq n$. This is the import of the next three lemmas.

Lemma 14 $\mathcal{F} \vdash^*$ *"If* $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ *is a* $\leq$*1-scarred subformula, if* $n - m = \Delta_{u+1}$*, if* $m < i \leq j \leq n$ *and if* $\mathbf{x}[\mathbf{j}]$ *is a unary connective* $\circledcirc$ *then* $Value_0(\mathbf{x}, [i,j], (m,n])$ *is equal to* $Value_0(\mathbf{x}, [i, j-1], (m,n]) \circ (s_1, s_2)$ *where* (s_1, s_2) *is the pair of Boolean truth values giving the truth value of the* $\leq$*1-scarred formula '*$\circledcirc$*'. As a special case, if* $i = j$ *then* $Value_0(\mathbf{x}, [i,j], (m,n])$ *is equal to* (s_1, s_2)*."*

There are four possible unary functions for $\circledcirc$; but generally, depending on the language, $\circledcirc$ will be negation ($\neg$) and then (s_1, s_2) will equal $(\bot, \top)$.

Lemma 15 $\mathcal{F} \vdash^*$ *"Suppose* $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ *is a* ≤ 1*-scarred subformula,* $n - m = \Delta_{u+1}$*,* $m < i \leq j \leq n$ *and* $\mathbf{x}[\mathbf{j}]$ *is a binary connective* $\circledcirc$ *and let* $f_{\circledcirc}$ *be the binary function such that*

$$f_{\circledcirc}((s_1, s_2), (t, t)) = (s_1 \circledcirc t, s_2 \circledcirc t).$$

Then

(a) *If* $\mathbf{x}[\mathbf{i}, \mathbf{j-1}]$ *is an unscarred formula,* $Value_0(\mathbf{x}, [i, j], (m, n])$ *is equal to*

$$f_{\circledcirc}((\top, \bot), Value_0(\mathbf{x}, [i, j-1], (m, n])).$$

(b) *Otherwise, let* $k \in [i, j]$ *be such that* $\mathbf{x}[\mathbf{k}, \mathbf{j-1}]$ *is a formula (unscarred). Then* $Value_0(\mathbf{x}, [i, j], (m, n])$ *is equal to*

$$f_{\circledcirc}\left(Value_0(\mathbf{x}, [i, k-1], (m, n]), Value_0(\mathbf{x}, [k, j-1], (m, n])\right)."$$

With our conventions for the truth of undefined subformulas, (a) is a special case of (b).

Proof (Outline) The $\mathcal{F}$-proofs for Lemmas 14 and 15 are again brute force "induction" proofs. For the proof of Lemma 14 the following stronger assertion is proved:

> "If $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ is a $\leq$1-scarred subformula and $1 \leq p_\ell \leq 4$ for each ℓ and if $Int_k((m, n], \vec{p})$ includes $\mathbf{x}[\mathbf{i}, \mathbf{j}]$ and if $\mathbf{x}[\mathbf{j}]$ is a unary operator $\circledcirc$ then
>
> $$Value_k(\mathbf{x}, [i, j], (m, n], \vec{p})$$
>
> is equal to
>
> $$Value_k(\mathbf{x}, [i, j-1], (m, n], \vec{p}) \circ (s_1, s_2)$$
>
> where (s_1, s_2) is the truth value of $\circledcirc$."

The quoted assertion is proved for all appropriate values of $i,\ j,\ p_1, \ldots, p_k$; first for $k = u+1$ then for $k = u,\ u-1$, etc., down to $k = 0$. For each k there are only polynomially many values for $i, j, \vec{p}$ and all the assertions for a given value of k may be readily proved by polynomial size $\mathcal{F}$-proofs from the assertions for $k+1$. These proofs from the assertions for $k+1$ yielding the assertions for k involve a finite number of cases depending on where the breakpoints fall. We leave the details to the reader.

The $\mathcal{F}$-proof of Lemma 15 proves a similarly generalized version of the lemma and also proceeds by brute force "induction" on k.

Corollary 16 $\mathcal{F} \vdash^* $ *"If* $\mathbf{x}[\mathbf{i},\mathbf{j}]$ *is a formula and if* $m_k < i \leq j \leq n_k$ *and* $n_k - m_k = \Delta_{u_k+1}$ *for* $k = 1, 2$ *then*

$$Value_0(\mathbf{x}, [i,j], (m_1, n_1]) = Value_0(\mathbf{x}, [i,j], (m_2, n_2]).$$

Corollary 16 follows froms Lemmas 14 and 15 by another brute force "induction" proof. The "induction" is on the length $1 + j - i$ of the formula and the argument hinges on the fact that $Value_0(\cdots)$ respects the meanings of the unary and binary propositional connectives.

In view of Corollary 16 we can define the notion of the truth of a formula independently of m and n. We let $TRUE(\mathbf{x}, [i,j])$ denote the polynomial size formula with variables $\mathbf{x}$ and with variables $p_1, \ldots$ which may be named in the formula coded by $\mathbf{x}$ such that $TRUE(\mathbf{x}, [i,j])$ is true if and only if the formula coded by $\mathbf{x}[\mathbf{i},\mathbf{j}]$ is true. The definition of $TRUE$ justs picks an arbitrary interval $(m, n]$ containing $[i, j]$ with $n - m = \Delta_{u+1}$ for some $u \geq 0$ and uses the formula $Value_0$.

Finally we a very useful lemma relating actual truth and truth as defined by $TRUE$. The lemma states that there are polynomial sized $\mathcal{F}$-proofs that if a formula is true according to the definition of $TRUE$ then it is in fact true. More precisely:

Lemma 17 *Let* φ *be a formula in the language of* $\mathcal{F}$*. Then*

$$\mathcal{F} \vdash^* (\text{“}\mathbf{x}[\mathbf{i},\mathbf{j}] \text{ encodes } \varphi\text{”} \rightarrow (TRUE(\mathbf{x}, [i,j])) \leftrightarrow \varphi)).$$

The $\mathcal{F}$-proof of this lemma is another brute force "induction" proof on the length of φ. Note that the assumption that φ is in the language of $\mathcal{F}$ is not superfluous since we have defined (in $\mathcal{F}$) the truth of formulas in the language of $\mathcal{F}_+$.

4 Applications of the Truth Definition

It is now easy to prove Theorems 1-3 using the $\mathcal{F}$-intensional polynomial size definition of truth of formulas.

Main Theorem 1 $\mathcal{F}_0 \overset{*}{\vdash} Con_{\mathcal{F}_+}(\mathbf{x})$. More generally, if $\mathcal{F}_1$ and $\mathcal{F}_2$ are Frege proof systems, then $\mathcal{F}_1 \overset{*}{\vdash} Con_{\mathcal{F}_2}(\mathbf{x})$.

Proof We start by proving that $\mathcal{F}_0 \overset{*}{\vdash} Con_{\mathcal{F}_+}(\mathbf{x})$. By Theorem 11 it suffices to show $\mathcal{F}_0 \overset{*}{\vdash} Con_{PLOF-\mathcal{F}_+}(\mathbf{x})$. So we suppose $\mathbf{x}$ codes a PLOF-$\mathcal{F}_+$-proof of $\bot$ and argue informally inside $\mathcal{F}_0$. Without loss of generality, no propositional variables occur in the proof coded by $\mathbf{x}$ since otherwise any variable p_j can be replaced everywhere by $\top$. Now, by brute force "induction" it is easy to prove that any formula, say $\mathbf{x}[\mathbf{i},\mathbf{j}]$, appearing as a line in the proof coded by $\mathbf{x}$ is true, i.e., $TRUE(\mathbf{x}[\mathbf{i},\mathbf{j}])$. This is proved successively for $j = 1, 2, \ldots$ and for all values $i \leq j$; each $\mathbf{x}[\mathbf{i},\mathbf{j}]$ is either an instance of an axiom scheme or is inferred by modus ponens and in either case, Lemmas 14 and 15 shows that $\mathbf{x}[\mathbf{i},\mathbf{j}]$ is true. But now Lemma 17 yields a contradiction since the last line of $\mathbf{x}$ is $\bot$ which is not true.

This argument shows that $\mathcal{F}_0 \overset{*}{\vdash} Con_{\mathcal{F}_2}(n)$ for any Frege system $\mathcal{F}_2$ whose language involves only connectives of arity less than or equal to two. For more general Frege systems with k-ary connectives a more sophisticated truth definition is needed; see section 6 of Buss [1] for the essential idea.

Finally, Reckhow [10] shows that any Frege system $\mathcal{F}_1$ p-simulates $\mathcal{F}_0$ by a simple "direct translation". A direct translation is a translation of $\mathcal{F}_0$-formulas into the language of $\mathcal{F}_1$ that gives only a linear increase in the size of the formulas. This hinges on the fact that if $\mathcal{L}_1$ is a truth functionally complete set of connectives then there are $\mathcal{L}_1$ formulas $\varphi_\neg(p_0)$ and $\varphi_\wedge(p_0, p_1)$ which are tautologically equivalent to $\neg p_0$ and to $p_0 \wedge p_1$ and such that p_0 occurs exactly once in $\varphi_\neg(p_0)$ and p_0 and p_1 each occur exactly once in $\varphi_\wedge(p_0, p_1)$. (However, $\varphi_\neg$ and $\varphi_\wedge$ may use multiple occurences of another propositional variable if there is no constant symbol $\top$ or $\bot$ in $\mathcal{L}_1$.) Using $\varphi_\neg$ and $\varphi_\wedge$, formulas involving $\neg$ and $\wedge$ may be translated into $\mathcal{L}_1$-formulas with a only a linear increase in size and this leads to a translation of $\mathcal{F}_0$-proofs into $\mathcal{F}_1$-proofs with only a linear increase in the size of a proof. Reckhow was the first to show that such direct translations exist, see Buss, et. al. [4] for another proof. $\square$

Main Theorem 2 (Reckhow [10]). $\mathcal{F}_0$ p-simulates $\mathcal{F}_+$. More generally, for any two Frege proof systems $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1$ p-simulates $\mathcal{F}_2$.

Proof We show $\mathcal{F}_0$ p-simulates $\mathcal{F}_+$. For this proof, we enlarge the language of $\mathcal{F}_0$ to include $\top$ and $\bot$ as unary constants; appropriate axioms are also added to keep $\mathcal{F}_0$ complete in the enlarged language. Adding these new constants does not change the lengths of proofs in any essential way since $\top$ and $\bot$ can be defined as $p_0 \vee \neg p_0$ and $p_0 \wedge \neg p_0$.

Suppose φ is an $\mathcal{F}_0$-formula with free variables $\vec{p}$ and that there is an $\mathcal{F}_+$-proof of φ of length m. Let $\mathbf{s}$ be a sequence of constants $\top$, $\bot$ which code the $\mathcal{F}_+$-proof. Without loss of generality, we may assume that the length of the sequence $\mathbf{s}$ is $O(m \cdot \log m)$ since, if necessary, we may rename variables so that the subscripts are $\leq m$. By the methods of section 2, there is a polynomial length $\mathcal{F}_0$-formula expressing *"*$\mathbf{s}$ *codes an* $\mathcal{F}_+$*-proof of* φ*"*; and since this is a true, variable-free formula (involving constants $\top$ and $\bot$), it has an $\mathcal{F}_0$-proof of length $O(p(m \log m))$ for some polynomial p. But now, as in the previous proof, $\mathcal{F}_0 \vdash TRUE(\mathbf{s}, [i, j])$ with proofs of length polynomial in m for all $\mathbf{s}[\mathbf{i}, \mathbf{j}]$ which code formulas. Now by Lemma 17, $\mathcal{F}_0$ proves φ with a proof of length polynomial in m.

The above shows that $\mathcal{F}_0$ simulates $\mathcal{F}_+$; by the uniformity of the definition of truth, $\mathcal{F}_0$ p-simulates $\mathcal{F}_+$. The same argument shows $\mathcal{F}_0$ p-simulates any Frege system $\mathcal{F}_2$.

As argued above, any Frege system p-simulates $\mathcal{F}_0$ by direct translations. It now follows that any two Frege systems p-simulate each other by the transitivity of p-simulation. □

Main Theorem 3 Frege proof systems p-simulate extended Frege proof systems if and only if $\mathcal{F} \overset{*}{\vdash} Con_{e\mathcal{F}}(\mathbf{x})$ (for $\mathcal{F}$ any Frege proof system).

Proof ($\Longrightarrow$) The forward implication follows immediately from the fact that $e\mathcal{F} \overset{*}{\vdash} Con_{e\mathcal{F}}(\mathbf{x})$ (Cook [5]).

($\Longleftarrow$) Suppose $\mathcal{F} \overset{*}{\vdash} Con_{e\mathcal{F}}(\mathbf{x})$. Let φ be a formula involving propositional variables $\vec{p}$ which has an $e\mathcal{F}$-proof of length m. We must show that φ has an $\mathcal{F}$-proof of length $\leq r(m)$ for some polynomial r. Let $\mathbf{s}$ again be a sequence of constants $\top$, $\bot$ coding the $e\mathcal{F}$-proof of φ. As in the previous proof, w.l.o.g. $|\mathbf{s}|_\Sigma < m \log m$ and there is a polynomial size (in m) $\mathcal{F}$-proof of *"*$\mathbf{s}$ *codes an* $e\mathcal{F}$*-proof of* φ*"*. For the rest of the proof, we argue informally with polynomial

size proofs in the theory $\mathcal{F}_0$. Let T_{p_i} be the nullary connective $\top$ if p_i is true and be the nullary connective $\bot$ if p_i is false. Suppose $\neg\varphi(\vec{p})$. Then there is an $\mathcal{F}$-proof of $\neg\varphi(\vec{T_p})$ of size $\leq p(|\varphi|_\Sigma)$ for some polynomial p.[10] Now modify the proof coded by $\mathbf{s}$ by replacing each occurence of any variable p_i by the constant T_{p_i}; this clearly yields an $e\mathcal{F}$-proof of $\varphi(\vec{T_p})$ which is also of size m.[11] Combining the $e\mathcal{F}$-proof of $\varphi(\vec{T_p})$ with the $\mathcal{F}$-proof of $\neg\varphi(\vec{T_p})$ easily yields an $e\mathcal{F}$-proof of "$\bot$" of size $\leq p(|\varphi|) + m + \alpha \cdot |\varphi|$ for some constant α. Thus we have shown (with a polynomial size $\mathcal{F}$-proof) that if $\neg\varphi$ then $\neg Con_{e\mathcal{F}}(p(|\varphi|) + (\alpha+1)m)$. But since $|\varphi| < m$ and by the hypothesis, there is a polynomial size $\mathcal{F}$-proof of $Con_{e\mathcal{F}}(p(|\varphi|) + (\alpha+1)m)$. Hence there is an $\mathcal{F}$-proof of φ of polynomial size in m. $\square$

References

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

[2] Samuel R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

[3] Samuel R. Buss, Steven A. Cook, Arvind Gupta, and Vijaya Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21:755–780, 1992.

[4] Samuel R. Buss and et al. Weak formal systems and connections to computational complexity. Student-written Lecture Notes for a Topics Course at U.C. Berkeley, January–May 1988.

[5] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97, 1975.

[10] This is by the formalization in $\mathcal{F}$ of an argument already employed in the previous proof. Namely, exhaustively show that for each subformula ψ of $\neg\varphi$, if $\psi(\vec{p})$ then there is an $\mathcal{F}_0$-proof of $\psi(\vec{T_p})$, and if $\neg\psi(\vec{p})$ then there is an $\mathcal{F}_0$-proof of $\neg\psi(\vec{T_p})$. These proofs are obviously polynomial size in the length of ψ.

[11] Since without loss of generality we can assume that no p_i appears in an axiom scheme.

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

[7] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first-order theories and the complexity of computations. *Journal of Symbolic Logic*, 54:1063–1079, 1989.

[8] Pavel Pudlák. On the lengths of proofs of finitistic consistency statements in first order theories. In *Logic Colloquium '84*, pages 165–196. North-Holland, 1986.

[9] Pavel Pudlák. Improved bounds to the lengths of proofs of finitistic consistency statements. In *Logic and Combinatorics*, volume 65 of *Contemporary Mathematics*, pages 309–331. American Mathematical Society, 1987.

[10] Robert A. Reckhow. *On the Lengths of Proofs in the Propositional Calculus*. PhD thesis, Department of Computer Science, University of Toronto, 1976. Technical Report #87.