On Interpolation and Automatization for Frege Systems

Maria Luisa Bonet [*] Toniann Pitassi [†] Ran Raz [‡]

Abstract

The interpolation method has been one of the main tools for proving lower bounds for propositional proof systems. Loosely speaking, if one can prove that a particular proof system has the feasible interpolation property, then a generic reduction can (usually) be applied to prove lower bounds for the proof system, sometimes assuming a (usually modest) complexity-theoretic assumption. In this paper, we show that this method cannot be used to obtain lower bounds for Frege systems, or even for TC^0-Frege systems. More specifically, we show that unless factoring (of Blum integers) is feasible, neither Frege nor TC^0-Frege has the feasible interpolation property. In order to carry out our argument, we show how to carry out proofs of many elementary axioms/theorems of arithmetic in polynomial size TC^0-Frege.

As a corollary, we obtain that TC^0-Frege as well as any proof system that polynomially simulates it, is not automatizable (under the assumption that factoring of Blum integers is hard). We also show under the same hardness assumption, that the k-provability problem for Frege systems is hard.

1 Introduction

One of the most important questions in propositional proof complexity is to show that there is a family of propositional tautologies requiring superpolynomial size proofs in a Frege or Extended Frege proof system. The problem is still open, and it is thus a very important question to understand which techniques can be applied to prove lower bounds for these systems, as well as for weaker systems. In recent years, the interpolation method has been one of the most promising approaches for proving lower bounds for propositional proof systems and for bounded arithmetic. Here we show that this method is not likely to work for Frege systems and some weaker systems. The basic idea behind the interpolation method is as follows.

[*]Department of LSI, Universidad Politécnica de Cataluña, Barcelona, Spain, bonet@lsi.upc.es. Research partly supported by EU HCM network console and by ESPRIT LTR Project no. 20244 (ALCOM-IT), and TIC98-0410-C02-01

[†]Department of Computer Science, University of Arizona, toni@cs.arizona.edu. Research supported by NSF Grant CCR-9457782, US-Israel BSF Grant 95-00238, and Grant INT-9600919/ME-103 from NSF and MŠMT (Czech Republic)

[‡]Department of Applied Math, Weizmann Institute, ranraz@wisdom.weizmann.ac.il. Research supported by US-Israel BSF Grant 95-00238

We begin with an unsatisfiable statement of the form $F(x, y, z) = A_0(x, z) \wedge A_1(y, z)$, where z denotes a vector of shared variables, and x and y are vectors of private variables for formulas A_0 and A_1 respectively. Since F is unsatisfiable, it follows that for any truth assignment α to z, either $A_0(x, \alpha)$ is unsatisfiable or $A_1(y, \alpha)$ is unsatisfiable. An interpolation function associated with F is a boolean function that takes such an assignment α as input, and outputs 0 only if A_0 is unsatisfiable, and 1 only if A_1 is unsatisfiable. (Note that both A_0 and A_1 can be unsatisfiable in which case either answer will suffice).

How hard is it to compute an interpolation function for a given unsatisfiable statement F as above ? It has been shown, among other things, that interpolation functions are not always computable in polynomial time unless $P = NP \cap co - NP$ [M1, M2, M3]. Nevertheless, it is possible that such a procedure exists for some special cases. In particular, a very interesting and fruitful question is whether one can find (or whether there exists) a polynomial size circuit for an interpolation function, in the case where F has a short refutation in some proof system S. We say that a proof system S admits *feasible interpolation* if whenever S has a polynomial size refutation of a formula F (as above), an interpolation function associated with F has a polynomial size circuit. Krajíček [K2] was the first to make the connection between proof systems having feasible interpolation and circuit complexity.

There is also a monotone version of the interpolation idea. Namely, for conjunctive normal form formulas A_0 and A_1, $F = A_0(x, z) \wedge A_1(y, z)$ is *monotone* if the variables of z occur only positively in A_1 and only negatively in A_0. In this case, an associated interpolant function is monotone, and we are thus interested in finding a polynomial size monotone circuit for an interpolant function. We say that a proof system S admits *monotone feasible interpolation* if whenever S has a polynomial size refutation of a monotone F, a monotone interpolation function associated with F has a monotone polynomial size circuit.

Beautiful connections exist between circuit complexity, and proof systems having feasible interpolation, in both (monotone and non-monotone) cases:

In the monotone case, superpolynomial lower bounds can be proven for a (sufficiently strong) proof system that admits feasible interpolation. This was presented by the sequence of papers [IPU, BPR, K1], and was first used in [BPR] to prove lower bounds for propositional proof systems. (The idea is also implicit in [Razb2]).

In short, the statement F that is used is the Clique interpolation formula, $A_0(g, x) \wedge A_1(g, y)$, where A_0 states that g is a graph containing a clique of size k (where the clique is described by the x variables), and A_1 states that g is a graph that can be colored with $k - 1$ colors (where the coloring is described by the y variables). By the pigeonhole principle, this formula is unsatisfiable. However, an associated monotone interpolation function would take as input a graph g, and distinguish between graphs containing cliques of size k from those that can be colored with $k - 1$ colors. By [Razb1, AB], when $k = n^{2/3}$, such a circuit is of exponential size. Thus, exponential lower bounds follow for any propositional proof system S that admits feasible monotone interpolation.

Similar ideas also work in the case where S admits feasible interpolation (but not necessarily monotone feasible interpolation). The first such result, by [Razb2], gives *explicit* superpolynomial lower bounds for (sufficiently strong) proof systems S admitting feasible interpolation, under a cryptographic assumption. In particular, it was shown that a (non-monotone) interpolation function, associated with a certain statement expressing $P \neq NP$, is computable by polynomial size circuits only if there do not exist pseudorandom number generators.

Therefore, lower bounds follow for any (sufficiently strong) propositional proof system that admits feasible interpolation (conditional on the cryptographic assumption that there exist pseudorandom number generators). It is also possible to prove nonexplicit superpolynomial lower bounds for a (sufficiently strong) proof system under the assumption that NP is not computable by polynomial sized circuits.

Many researchers have used these ideas to prove lower bounds for propositional proof systems. In particular, in the last five years, lower bounds have been shown for all of the following systems using the interpolation method: Resolution [BPR], Cutting Planes [IPU, BPR, Pud, CH], generalizations of Cutting Planes [BPR, K1, K3], relativized bounded arithmetic [Razb2], Hilbert's Nullstellensatz [PS], the polynomial calculus [PS], and the Lovasz-Schriver proof system [Pud3].

1.1 Automatizability and k-provability

As explained in the previous paragraphs, the existence of feasible interpolation for a particular proof system S gives rise to lower bounds for S. Feasible interpolation, moreover, is a very important paradigm for proof complexity (in general) for several other reasons. In this section, we wish to explain how the lack of feasible interpolation for a particular proof system S implies that S is not automatizable.

We say that a proof system S is *automatizable* if there exists a deterministic procedure D that takes as input a formula f and returns an S-refutation of f (if one exists) in time polynomial in the size of the shortest S-refutation of f. Automatizability is a crucial concept for automated theorem proving: in proof complexity we are mostly interested in the length of the shortest proof, whereas in theorem proving it is also essential to be able to find the proof. While there are seemingly powerful systems for the propositional calculus (such as Extended Resolution or even ZFC), they are scarce in theorem proving because it seems difficult to search efficiently for a short proof in such systems. In other words, there seems to be a tradeoff between proof simplicity and automatizability – the simpler the proof system, the easier it is to find the proof.

In this section, we formalize this tradeoff in a certain sense. In particular, we show that if S has no feasible interpolation then S is not automatizable. This was first observed by Russell Impagliazzo. The idea is to show that if S is automatizable (using a deterministic procedure D), then S has feasible interpolation.

Theorem 1 *If a proof system S does not have feasible interpolation, then S is not automatizable.*

Proof Suppose that S is automatizable, and suppose D is the deterministic procedure to find proofs, and moreover, D is guaranteed to run in time n^c, where n is the size of the shortest proof of the input formula. Let $A_0(x,z) \wedge A_1(y,z)$ be the interpolant statement, and let α be an assignment to z. We want output an interpolant function for $A_0(x,\alpha) \wedge A_1(y,\alpha)$. First, we run D on $A_0(x,z) \wedge A_1(y,z)$ to obtain a refutation of size s. Next, we simulate D on $A_0(x,\alpha)$ for $T(s)$ steps, and return 0 if and only if D produces a refutation of $A_0(x,\alpha)$ within time $T(s)$. $T(s)$ will be chosen to be the maximum time for D to produce a refutation for a formula that has a refutation of size s; thus $T(s) = s^c$ in this case. This works because in the case where $A_1(y, \alpha)$ is satisfiable with satisfying assignment γ, we can plug γ into the refutation of $A_0(x, \alpha) \wedge A_1(y, \alpha)$ to obtain a refutation of $A_0(x, \alpha)$ of size s. Therefore S has feasible interpolation. $\square$

Thus, feasible interpolation is a simple measure that formalizes the complexity/search trade-off: the existence of feasible interpolation implies superpolynomial lower bounds (sometimes modulo complexity assumptions), whereas the nonexistence of feasible interpolation implies that the proof system cannot be automatized.

A concept that is very closely related to automatizability is k-provablity. The k-symbol provability problem for a particular Frege system S is as follows. The problem is to determine, given a propositional formula f and a number k, whether or not there is a k-symbol S proof of f. The k-line provability problem for S is to determine whether or not there is a k-line S proof of f. The k-line provability is an undecidable problem for first-order logic [B1]; and the first complexity result for the k-provability problem for propositional logic was provided by Buss [B2] who proved the rather surprising fact that the k-symbol propositional provability problem is NP-complete for a particular Frege system. More recently, [ABMP] show that the k-symbol and k-line provability problems cannot be approximated to within linear factors for a variety of propositional proof systems, including Resolution and all Frege systems, unless $P = NP$.

The methods in our paper show that both the k-symbol and k-line provability problems cannot be solved in polynomial time for any TC^0-Frege system, Frege system, or Extended Frege system, assuming hardness of factoring (of Blum integers). More precisely, using the same idea as above, we can show that if there is a polynomial time algorithm A solving the k-provability problem for S, then S has feasible interpolation: Suppose that $F = A_0(x, z) \wedge A_1(y, z)$ is the unsatisfiable statement. We first run A with $k = n, n^2, n^3, \ldots$ on F, until A first verifies that there is a size $s = |F|^c$ proof of F for some fixed value of c. Now let α be an assignment to z. As above, we run A to determine if there is an $O(s)$-symbol (or $O(s)$-line) refutation of $A_0(x, \alpha)$ and return 0 if and only if A accepts. In fact, this proof can be extended easily to show that both the k-symbol and k-line provability problems cannot be approximated to within polynomial factors for the same proof systems (TC^0-Frege, Frege, Extende Frege) under the same hardness assumption.

1.2 Interpolation and one way functions

How can one prove that a certain propositional proof system S does not admit feasible interpolation ? One idea, due to Krajíček and Pudlák [KP], is to use one way permutations in the following way. Let h be a one way permutation and let $A_0(x, z), A_1(y, z)$ be the following formulas.

The formula A_0 :
$h(x) = z$, AND the i^{th} bit of x is 0.

The formula A_1 :
$h(y) = z$, AND the i^{th} bit of y is 1.

Since h is one to one, $A_0(x, z) \wedge A_1(y, z)$ is unsatisfiable. Assume that A_0, A_1 can be formulated in the proof system S, and that in S there exists a polynomial size refutation for

$A_0(x,z) \wedge A_1(y,z)$. Then, if S admits feasible interpolation it follows that given an assignment α to z there exists a polynomial size circuit that decides whether $A_0(x,\alpha)$ is unsatisfiable or $A_1(y,\alpha)$ is unsatisfiable. Obviously, such a circuit breaks the i^{th} bit of the input for h. Since A_0, A_1 can be constructed for any i, all bits of the input for h can be broken. Hence, assuming that the input for h is secure, and that in the proof system S there exists a polynomial size refutation for $A_0 \wedge A_1$, it follows that S does not admit feasible interpolation.

A major step towards the understanding of feasible interpolation was made by Krajíček and Pudlák [KP]. They considered formulas A_0, A_1 based on the RSA cryptographic scheme, and showed that unless RSA is not secure, Extended Frege systems do not have feasible interpolation. It has been open, however, whether or not the same negative results hold for Frege systems, and for weaker systems such as bounded depth threshold logic or bounded depth Frege.

1.3 Our results

In this paper, we prove that Frege systems, as well as constant-depth threshold logic (referred to below as TC^0-Frege), do not admit feasible interpolation, unless factoring of Blum integers is computable by polynomial size circuits. (Recall that Blum integers are integers P of the type $P = p_1 \cdot p_2$, where p_1, p_2 are both primes such that $p_1 \bmod 4 = p_2 \bmod 4 = 3$.) Thus our result significantly extends [KP] to weaker proof systems. In addition, our cryptographic assumption is weaker.

To prove our result, we use a variation of the ideas of [KP]. In a conversation with Moni Naor, he observed that the cryptographic primitive needed here is not a one way permutation as in [KP], but the more general structure of *bit commitment.* Our formulas A_0, A_1 are based on the Diffie-Hellman secret key exchange scheme [DH]. For simplicity, we state the formulas only for the least significant bit. (Our argument works for any bit).

Informally, our propositional statement DH will be

$$DH_n = A_0(P,g,X,Y,a,b) \wedge A_1(P,g,X,Y,c,d).$$

The common variables are two integers X, Y, and P and g. P represents a number (not necessarily a prime) of length n, and g an element of the group Z_P^*. The private variables for A_0 are integers a, b, and the private variables for A_1 are integers c, d.

Informally, $A_0(P,g,X,Y,a,b)$ will say that $g^a \bmod P = X$, $g^b \bmod P = Y$, and that $g^{ab} \bmod P$ is even. Similarly, $A_1(P,g,X,Y,c,d)$ will say that $g^c \bmod P = X$, $g^d mod P = Y$, and $g^{cd} \bmod P$ is odd. The statement $A_0 \wedge A_1$ is unsatisfiable since (informally) if A_0, A_1 are both true we have

$$g^{ab} \bmod P = (g^a \bmod P)^b \bmod P = X^b \bmod P =$$

$$(g^c \bmod P)^b \bmod P = g^{bc} \bmod P = (g^b \bmod P)^c \bmod P =$$

$$Y^c \bmod P = (g^d \bmod P)^c \bmod P = g^{cd} \bmod P.$$

We will show that the above informal proof can be made formal with a (polynomial size) TC^0-Frege proof. On the other hand, an interpolant function computes one bit of the secret key exchanged by the Diffie-Hellman procedure. Thus, if TC^0-Frege admits feasible interpolation, then all bits of the secret key exchanged by the Diffie-Hellman procedure can be broken using polynomial size circuits, and hence the Diffie-Hellman cryptographic scheme is not secure. Note, that it was proved that for $P = p_1 \cdot p_2$, where p_1, p_2 are both primes such that $p_1 \bmod 4 = p_2 \bmod 4 = 3$ (i.e., P is Blum integer), breaking the Diffie-Hellman cryptographic scheme is harder than factoring P ! [BBR] (see also [Sh, Mc]).

It will require quite a bit of work to formalize the above statement and argument with a short TC^0-Frege proof. Notice that we want the size of the propositional formula expressing the Diffie Hellman statement to be polynomially bounded in the number of binary variables. And additionally, we want the size of the TC^0-Frege proof of the statement to also be polynomially bounded. A key idea in order to define the statement and prove it efficiently, is to introduce additional common variables to our propositional Diffie-Hellman statement. The bulk of the argument then involves showing how (with the aid of the auxillary variables) one can formalize the above proof by showing that basic arithmetic facts, including the Chinese Remainder Theorem, can be stated and proven efficiently within TC^0-Frege.

1.4 Section description

The paper is organized as follows. In Section 2, we define our TC^0-Frege system. In Section 3, we define the TC^0-formulas used for the proof. In Section 4, we define precisely the interpolation formulas which are based on the Diffie-Hellman cryptographic scheme. In Section 5 we show how to prove our main theorem, provided we have some technical lemmas that will be proved fully in section 7. In Section 6 there is a discussion and some open problems. Finally in section 7 we prove all the technical lemmas required for the main theorem.

The unusual organization of the paper is due to the many very technical lemmas required to show the result, that are essential to the correctness of the argument, but not every reader might want to go through. Sections 1-6 give an exposition of the result, relying on the complete proofs in the technical part.

2 TC^0-Frege systems

For clarity, we will work with a specific bounded-depth threshold logic system, that we call TC^0-Frege. However, any reasonable definition of such a system should also suffice. Our system is a sequent-calculus logical system where formulas are built up using the connectives $\vee$, $\wedge$, Th_k, $\neg$, $\oplus_0$ and $\oplus_1$. ($Th_k(x)$ is true if and only if the number of 1's in x is at least k, and $\oplus_i(x)$ is true if and only if the number of 1's in x is $i \bmod 2$.)

Our system is essentially the one introduced in [MP]. (Which is, in turn, an extension of the system PTK introduced by Buss and Clote [BC, Section 10].)

Intuitively, a family of formulas $f_1, f_2, f_3, \ldots$ has polynomial size TC^0-Frege proofs if each formula has a proof of size polynomial in the size of the formula, and such that every line in the proof is a TC^0 formula.

Definition 2 *Formulas are built up using the connectives* $\wedge$, $\vee$, Th_k, $\oplus_1$, $\oplus_0$, $\neg$. *All connectives are assumed to have unbounded fan-in.* $\mathrm{Th}_k(A_1, \ldots, A_n)$ *is interpreted to be true if* *and only if the number of true A_i's is at least k; $\oplus_j(A_1, ..., A_n)$ is interpreted to be true if and only if the number of true A_i's is equal to $j \mod 2$.*

The formula $\wedge(A_1, \ldots, A_n)$ denotes the logical AND of the multi-set consisting of $A_1, \ldots A_n$, and similarly for $\vee$, $\oplus_j$ and Th_k. Thus commutativity of the connectives is implicit. Our proof system operates on sequents which are sets of formulas of the form $A_1, \ldots, A_i \rightarrow B_1, \ldots, B_j$. The intended meaning is that the conjunction of the A_i's implies the disjunction of the B_j's. A proof of a sequent S in our logic system is a sequence of sequents, $S_1, ..., S_q$, such that each sequent S_i is either an initial sequent, or follows from previous sequents by one of the rules of inference, and the final sequent, S_q, is S. The size of the proof is $\sum_{1 \leq i \leq q} \mathrm{size}(S_i)$ and its depth is $\max_{1 \leq i \leq q}(\mathrm{depth}(S_i))$.

The *initial sequents* are of the form: (1) $A \rightarrow A$ where A is any formula; (2) $\rightarrow \wedge()$; $\vee() \rightarrow$; (3) $\oplus_1() \rightarrow$; $\rightarrow \oplus_0()$; and (4) $\mathrm{Th}_k() \rightarrow$ for $k \geq 1$; $\rightarrow \mathrm{Th}_0(A_1, \ldots, A_n)$ for $n \geq 0$. The rules of inference are as follows. Note that the logical rules are defined for $n \geq 1$ and $k \geq 1$. First we have simple structural rules such as weakening (formulas can always be added to the left or to the right), contraction (two copies of the same formula can be replaced by one), and permutation (formulas in a sequent can be reordered). The remaining rules are the cut rule, and logical rules which allow us to introduce each connective on both the left side and the right side. The cut rule allows the derivation of $\Gamma, \Gamma' \rightarrow \Delta, \Delta'$ from $\Gamma, A \rightarrow \Delta$, and $\Gamma' \rightarrow A, \Delta'$.

The logical rules are as follows.

1. (Negation-left) From $\Gamma \rightarrow A, \Delta$, we can derive $\neg A, \Gamma \rightarrow \Delta$.
2. (Negation-right) From $A, \Gamma \rightarrow \Delta$, derive $\Gamma \rightarrow \neg A, \Delta$.
3. (And-left) From $A_1, \wedge(A_2, \ldots, A_n), \Gamma \rightarrow \Delta$ derive $\wedge(A_1, \ldots, A_n), \Gamma \rightarrow \Delta$.
4. (And-right) From $\Gamma \rightarrow A_1, \Delta$ and $\Gamma \rightarrow \wedge(A_2, \ldots, A_n), \Delta$ derive $\Gamma \rightarrow \wedge(A_1, \ldots, A_n), \Delta$
5. (Or-left) From $A_1, \Gamma \rightarrow \Delta$ and $\vee(A_2, \ldots, A_n), \Gamma \rightarrow \Delta$ derive $\vee(A_1, \ldots, A_n), \Gamma \rightarrow \Delta$
6. (Or-right) From $\Gamma \rightarrow A_1, \vee(A_2, \ldots, A_n), \Delta$ derive $\Gamma \rightarrow \vee(A_1, \ldots, A_n), \Delta$.
7. (Mod-left) From $A_1, \oplus_{1-i}(A_2, \ldots, A_n), \Gamma \rightarrow \Delta$ and $\oplus_i(A_2, \ldots, A_n), \Gamma \rightarrow A_1, \Delta$ derive $\oplus_i(A_1, \ldots, A_n),\ \Gamma \rightarrow \Delta$.
8. (Mod-right) From $A_1, \Gamma \rightarrow \oplus_{1-i}(A_2, \ldots, A_n), \Delta$ and $\Gamma \rightarrow A_1, \oplus_i(A_2, \ldots, A_n), \Delta$ derive $\Gamma \rightarrow \oplus_i(A_1, \ldots, A_n), \Delta$.
9. (Threshold-left) From $\mathrm{Th}_k(A_2, \ldots, A_n), \Gamma \rightarrow \Delta$ and $A_1, \mathrm{Th}_{k-1}(A_2, \ldots, A_n), \Gamma \rightarrow \Delta$ derive
   $\mathrm{Th}_k(A_1, \ldots, A_n), \Gamma \rightarrow \Delta$.
10. (Threshold-right) From $\Gamma \rightarrow A_1, \mathrm{Th}_k(A_2, \ldots, A_n), \Delta$ and $\Gamma \rightarrow \mathrm{Th}_{k-1}(A_2, \ldots, A_n), \Delta$ derive $\Gamma \rightarrow \mathrm{Th}_k(A_1, \ldots, A_n), \Delta$.

A TC^0 proof is a bounded-depth proof in our system of polynomial size. More formally we have the following definitions.

Definition 3 *Let $F = \{(\Gamma_n \to \Delta_n) : n \in N\}$ be a family of sequents. Then $\{R_n : n \in \mathbf{N}\}$ is a family of TC^0 proofs for F if there exist constants c and d such that the following conditions hold: (1) Each R_n is a valid proof of $(\Gamma_n \to \Delta_n)$ in our system, and 2) For all i, the depth of R_n is at most d; and (3) For all n, the size of R_n is at most $(\text{size}(\Gamma_n \to \Delta_n))^c$.*

We note that we have defined a specific proof system for clarity; our result still holds for any reasonable definition of a TC^0-Frege proof (it can be shown that our system polynomially simulates any Frege-style system). The difference between a polynomial size proof in our system and a polynomial size TC^0 proof is similar to the difference between NC^1 and TC^0.

3 The TC^0-formulas

In this section we will describe some of the TC^0-formulas needed to formulate and to refute the Diffie-Hellman formula. For simplicity of the description, let us assume that we have a fixed number N which is an upper bound for the **length** of all numbers used in the refutation of the Diffie-Hellman formula. The number N will be used to define some of the formulas below. After seeing the statement and the refutation of the Diffie-Hellman formula, it will be clear that it is enough to take N to be a small polynomial in the **length** of the number P used for the Diffie-Hellman formula.

3.1 Addition and subtraction

We will use the usual carry-save AC^0-formulas to add two n-bit numbers. Let $x = x_n, ..., x_1$ and $y = y_n, ..., y_1$ be two numbers. Then $x + y$ will denote the following AC^0-formula: There will be $n+1$ output bits, $z_{n+1}, ..., z_1$. The bit z_i will equal the mod 2 sum of C_i, x_i and y_i, where C_i is the carry bit. Intuitively, C_i is 1 if there is some bit position less than i that generates a carry that is propagated by all later bit positions until bit i. Formally, C_i is computed by $OR(R_{i(i-1)}, ..., R_{i1})$, where $R_{ij} = AND(P_{i-1}, ..., P_{j+1}, G_j)$, where $P_k = Mod_2(x_k, y_k)$, and $G_j = AND(x_j, y_j)$. (G_j is 1 if the j^{th} bit position generates a carry, and P_k is 1 if the k^{th} bit position propagates but does not generate a carry.)

As for subtraction, let us show how to compute $z = |x-y|$. Think of x, y as N-bit numbers. Let $s = x + \overline{y} + 1$, and similarly let $t = y + \overline{x} + 1$, where $\overline{y}$ is the complement (modulo 2) of the N bits of y, and $\overline{x}$ is the complement of the N bits of x. Denote $s = s_{N+1}, s_N, ..., s_1$, and note that s is equal to $2^N + (x-y)$, and similarly t is equal to $2^N + (y-x)$. If $s_{N+1} = 1$, then we know that $x - y \geq 0$ and thus $s = z$. Otherwise, if $s_{N+1} = 0$, then we know that $y - x > 0$ and thus $t = z$. Thus, for any i, we can compute z_i by $(s_{N+1} \wedge s_i) \vee (\neg s_{N+1} \wedge t_i)$.

3.2 Iterated addition

We will now describe the TC^0-formula $SUM[x_1, ..., x_m]$ that inputs m numbers, each n bits long, and outputs their sum $x_1 + x_2 + ... + x_m$ (see [CSV]). We assume that $m \leq N$. The main idea is to reduce the addition of m numbers to the addition of two numbers. Let x_i be $x_{i,n}, ..., x_{i,1}$ (in binary representation). Let $l = \lceil log_2 N \rceil$. Let $r = \frac{n}{2l}$, and assume (for simplicity) that r is an integer.

Divide each x_i into r blocks where each block has $2l$ bits, and let $S_{i,k}$ be the number in the k^{th} block of x_i. That is,

$$S_{i,k} = \sum_{j=1}^{2l} x_{i,(k-1)\cdot 2l+j} \cdot 2^{j-1}.$$

Now, each $S_{i,k}$ has $2l$ bits. Let $L_{i,k}$ be the low-order half of $S_{i,k}$ and let $H_{i,k}$ be the high order half. That is, $S_{i,k} = H_{i,k} \cdot 2^l + L_{i,k}$.

Denote

$$H = \sum_{i=1}^{m} \sum_{k=1}^{r} H_{i,k} \cdot 2^l \cdot 2^{(k-1)2l},$$

$$L = \sum_{i=1}^{m} \sum_{k=1}^{r} L_{i,k} \cdot 2^{(k-1)2l}.$$

Then,

$$x_1 + \ldots + x_m = \sum_{i=1}^{m} \sum_{k=1}^{r} S_{i,k} \cdot 2^{(k-1)2l} =$$

$$\sum_{i=1}^{m} \sum_{k=1}^{r} H_{i,k} \cdot 2^l \cdot 2^{(k-1)2l} + \sum_{i=1}^{m} \sum_{k=1}^{r} L_{i,k} \cdot 2^{(k-1)2l} = H + L.$$

Hence, we just have to show how to compute the numbers H, L. Let us show how to compute L, the computation of H is similar.

Denote $L_k = \sum_{i=1}^{m} L_{i,k}$. Then

$$L = \sum_{k=1}^{r} L_k \cdot 2^{(k-1)2l}.$$

Since each $L_{i,k}$ is of length l, each L_k is of length at most $l + \log_2 m$, which is at most $2l$. Hence, the bits of L are just the bits of the L_k's combined. That is, $L = L_r, L_{r-1}, \ldots, L_1$.

As for the computation of the L_k's, note that since each L_k is a poly-size sum of logarithmic length numbers, it can be computed using poly-size threshold gates.

3.3 Modular arithmetic

Next, we describe our TC_0-formulas that compute the quotient and remainder of a number z modulo p, where z is of length n. remainder and The inputs for the remainder and the quotient formulas are as follows:

1. the number z;
2. numbers $p_1, p_2, \ldots, p_n$;
3. numbers k_i and r_i for all $1 \leq i \leq n$.

The intended values for the variables k_i and r_i are such that $2^i = p \cdot k_i + r_i$, where $0 \leq r_i < p$, (for all $1 \leq i \leq n$). The intended values for the variables p_i are $i \cdot p$.

Suppose that $z = kp + r$, where $0 \leq r < p$, and assume that the input variables k_i, r_i and p_i take the right values. Then our formula $[z]_p$ will output r, and our formula $div_p(z)$ will output k. The formulas are computed as follows.

Let $z = z_n, ..., z_1$; i.e., $z = \sum_{i=1}^{n} 2^{i-1} z_i$. Suppose that the k_i, r_i and p_i variables satisfy $2^i = k_i \cdot p + r_i$, where $0 \leq r_i < p$, and $p_i = i \cdot p$ (for all $1 \leq i \leq n$). Then z satisfies

$$z = \sum_{i=1}^{n} 2^{i-1} z_i = \sum_{i=1}^{n} (p \cdot k_{i-1} + r_{i-1}) z_i =$$

$$p \cdot \sum_{i=1}^{n} k_{i-1} \cdot z_i + \sum_{i=1}^{n} r_{i-1} \cdot z_i.$$

Denote $s = \sum_{i=1}^{n} r_{i-1} \cdot z_i$, and let l be such that $l \cdot p \leq s < (l+1) \cdot p$. Then $[z]_p = s - l \cdot p$, and can therefore be computed by

$$[z]_p = SUM_{i=1}^{n}[r_{i-1} \cdot z_i] - p \cdot l.$$

$div_p(z)$ is computed by $SUM_{i=1}^{n}[k_{i-1} \cdot z_i] + l$.

Notice that if the k_i, r_i and p_i's are not such that $2^i = k_i \cdot p + r_i$, $0 \leq r_i < p$ and $p_i = i \cdot p$ then the formulas are not required to compute the correct values of the quotient or remainder, and can give junk.

3.4 Product and iterated product

We will write $x \cdot y$ to denote the formula $SUM_{i,j}[2^{i+j-2} x_i y_j]$, computing the product of two n-bit numbers x and y. By $2^{i+j-2} x_i y_j$ we mean 2^{i+j-2} if both x_i and y_j are true, and 0 otherwise.

Lastly, we will describe our TC^0-formula for computing the iterated product of m numbers. This formula is basically the original formula of [BCH], and articulated as a TC^0-formula in [M].

The iterative product, $PROD[z_1, ..., z_m]$ gives the product of $z_1, ..., z_m$, where each z_i is of length n, and we assume that m, n are both bounded by N. The basic idea is to compute the product modulo small primes using iterated addition, and then to use the constructive Chinese Remainder Theorem to construct the actual product from the product modulo small primes.

Let Q be the product of the first t primes $q_1, ..., q_t$, where t is the first integer that gives a number Q of length larger than N^2. Since $q_1, ..., q_t$ are all larger than 2, t is at most N^2, and by the well known bounds for the distribution of prime numbers the length of each q_j is at most $O(\log N)$. For each q_j, let g_j be a fixed generator for $Z^*_{q_j}$. Also, for each q_j, let $u_j \leq Q$ be a fixed number with the property that $u_j \bmod q_j = 1$ and for all $i \neq j$, $u_j \bmod q_i = 0$ (such a number exists by the Chinese Remainder Theorem). $PROD[z_1, ..., z_m]$ is computed as follows.

1. First we compute $r_{i,j} = [z_i]_{q_j}$, for all i, j. This is calculated using the modular arithmetic described earlier.

2. For each $1 \leq j \leq t$ we will compute $r_j = (\prod_{i=1}^{m} r_{i,j}) \bmod q_j$ as follows.

   a. Compute a_{ij} such that $(g_j^{a_{ij}}) \bmod q_j = r_{i,j}$. This is done by a table lookup.

   b. Calculate $c_j = SUM_{i=1}^{m}[a_{ij}]_{(q_j-1)}$.

   c. Compute r_j such that $g_j^{c_j} \bmod q_j = r_j$. This is another table lookup.

3. Finally compute

$$PROD[z_1, ..., z_m] = SUM_{j=1}^{t}[u_j \cdot r_j]_Q.$$

   We will hardwire the values $u_j \cdot k$ for all $k \leq q_j$. Thus, this computation is obtained by doing a table lookup to compute $u_j \cdot r_j$ followed by an iterated sum followed by a mod Q calculation.

3.5 Equality, and inequality

Often we will write $x = y$, where x and y are both vectors of variables or formulas: $x = x_1, ..., x_n$, and $y = y_1, .., y_n$. When we write $x = y$, we mean the formula $\wedge_i((\neg x_i \vee y_i) \wedge (x_i \vee \neg y_i))$. We apply the same conventions when writing $\neq, <, \leq, >, \geq$.

4 The Diffie-Hellman Formula

We are now ready to formally define our propositional statement DH. DH will be the conjunction of A_0 and A_1. The common variables for the formulas will be:

a). P and g representing n-bit integers, and for every $i \leq 2n$, we will also add common variables for g^{2^i} mod P.

b). X, Y, and for every $i \leq 2n$, we will also add common variables for X^{2^i} mod P, and for Y^{2^i} mod P.

c). We also add variables for $P_2, \ldots, P_N$, and for $k_1, \ldots, k_N$ and $r_1, \ldots, r_N$. These variables are needed to define arithmetic modulo P (see section 3.3).

For $e \in \{0, 1\}$, denote by $g^{2^i \cdot e}$ (respectively, $X^{2^i \cdot e}$, $Y^{2^i \cdot e}$) the following: the common variable g^{2^i} mod P (respectively X^{2^i} mod P, Y^{2^i} mod P) if $e = 1$, and 1 if $e = 0$. The formula $A_0(P, g, X, Y, a, b)$ will be the conjunction of the following TC^0-formulas:

1.

$$PROD_i \left[g^{2^i \cdot a_i}\right]_P = X.$$

   (Which means g^a mod $P = X$.)

2. For every $j \leq n$,

$$PROD_i \left[g^{2^{i+j} \cdot a_i}\right]_P = X^{2^j} \bmod P.$$

   (Which means $(g^{2^j})^a modP = X^{2^j} modP$.) Note that from this it is easy to prove for $e \in \{0, 1\}$,

$$PROD_i \left[g^{2^{i+j} \cdot a_i \cdot e}\right]_P = X^{2^j \cdot e}.$$

3. Similar formulas for $g^b \bmod P = Y$, and for $(g^{2^j})^b \bmod P = Y^{2^j} \bmod P$.
4. $PROD_{i,j}\left[g^{2^{i+j}\cdot a_i \cdot b_j}\right]_P$ is even. (Which means $g^{ab} \bmod P$ is even.)
5. For every $i \leq N$, formulas expressing $2^i = P \cdot k_i + r_i$, $1 \leq r_i < P$, and $P_i = i \cdot P$. (These formulas are added to guarantee that the modulo P arithmetic is computed correctly).

Similarly, the formula $A_1(P, g, X, Y, c, d)$ will be the conjunction of the above formulas, but with a replaced by c, b replaced by d, and the fourth item stating that $g^{cd} \bmod P$ is odd.

Note that the definition of the iterated product ($PROD$) requires the primes $q_1, ..., q_t$ (as well as their product Q, and the numbers $u_1, ..., u_t$), which are fixed for the length n. So we are going to hardwire the numbers $q_1, ..., q_t, Q, u_1, ... u_t$, as well as the correct values for the r_i's and k_i's needed for the modulo q_j arithmetic, for each one of these numbers.

5 A TC^0-Frege refutation for DH

We want to describe a TC^0-Frege refutation for DH. As mentioned above the proof proceeds as follows.

1. Using A_0, show that $g^{ab} \bmod P = X^b \bmod P$.
2. Using A_1, show that $X^b \bmod P = g^{cb} \bmod P$.
3. Show that $g^{cb} \bmod P = g^{bc} \bmod P$.
4. Using A_0, show that $g^{bc} \bmod P = Y^c \bmod P$.
5. Using A_1, show that $Y^c \bmod P = g^{dc} \bmod P$.
6. Show that $g^{dc} \bmod P = g^{cd} \bmod P$.

We can conclude from the above steps that A_0 and A_1 imply that $g^{ab} \bmod P = g^{cd} \bmod P$, but now we can reach a contradiction since A_0 states that $g^{ab} \bmod P$ is even, while A_1 states that $g^{cd} \bmod P$ is odd.

We formulate $g^{ab} \bmod P$ as

$$PROD_{i,j}\left[g^{2^{i+j}\cdot a_i \cdot b_j}\right]_P,$$

and $X^b \bmod P$ as

$$PROD_j\left[X^{2^j \cdot b_j}\right]_P.$$

Thus, step 1 is formulated as

$$PROD_{i,j}\left[g^{2^{i+j}\cdot a_i \cdot b_j}\right]_P = PROD_j\left[X^{2^j \cdot b_j}\right]_P,$$

and so on.

Steps 1,2,4,5 are all virtually identical. Steps 3 and 6 follow easily because our formulas defining g^{ab} make symmetry obvious. Thus the key step is to show step 1; that is, to show how to prove $g^{ab} \bmod P = X^b \bmod P$. As mentioned above, this is formulated as follows.

$$PROD_{i,j}\left[g^{2^{i+j}\cdot a_i\cdot b_j}\right]_P = PROD_j\left[X^{2^j\cdot b_j}\right]_P.$$

We will build up to the proof that $g^{ab} \bmod P$ equals $X^b \bmod P$ by proving many lemmas concerning our basic TC^0-formulas. The final lemma that we need is the following:

Lemma 4 *For every $z_{1,1}, ..., z_{m,m'}$ and p, there are TC^0-Frege proofs of*

$$PROD_{i,j}[z_{i,j}]_p = PROD_i[PROD_j[z_{i,j}]_p]_p.$$

The proof of the lemma is given in section 7.

Using Lemma 4 for the first equality, and point 2 from section 4 for the second equality, we can now obtain:

$$PROD_{i,j}\left[g^{2^{i+j}\cdot a_i\cdot b_i}\right]_P =$$
$$PROD_j\left[PROD_i\left[g^{2^{i+j}\cdot a_i\cdot b_j}\right]_P\right]_P = PROD_j\left[X^{2^j\cdot b_j}\right]_P,$$

which proves step 1.

The main goal of Section 7 is hence to show that the statement

$$PROD_{i,j}[z_{i,j}]_p = PROD_i[PROD_j[z_{i,j}]_p]_p,$$

has a short TC^0-Frege proof. This is not trivial because our TC^0-Frege formulas are quite complicated (and in particular the formulas for iterated product and modular arithmetic). In order to prove the statement, we will need to carry out a lot of the basic arithmetic in TC^0-Frege. Before we go on to the technical part, we will try to give some intuition on how the proof of the main lemma is built.

We organized the proof as a sequence of lemmas that show how many basic facts of arithmetic can be formulated and proved in TC^0-Frege (using our TC^0 formulas). The proofs of these lemmas require careful analysis of the exact formula used for each operation. The proof of some of these lemmas is straightforward (using the well known TC^0 formulas), while the proof of other lemmas require some new tricks.

In short, the main lemmas that are used for the TC^0-Frege proof of the final statement (Lemma 4) are the following:

1. (Lemma 38). For every x, y and p, there are TC^0-Frege proofs of

$$[x \cdot y]_p = [x \cdot [y]_p]_p.$$

2. (Lemma 41). For every $z_1, ..., z_m$, and every $1 \le k \le m$, there are TC^0-Frege proofs of

$$PROD[z_1, ..., z_m] = PROD[z_1, ..., z_{k-1}, PROD[z_k, ..., z_m]].$$

3. (Lemma 47). For every z_1, z_2, there are TC^0-Frege proofs of

$$PROD[z_1, z_2] = z_1 \cdot z_2.$$

First, we prove some basic lemmas about addition, subtraction, multiplication, iterative-sum, less-than, and modular arithmetic. Among theses lemmas will be Lemma 38.

The proof of Lemma 41 is cumbersome, but it is basically straightforward, given some basic facts about the modulo. Recall that to do the iterated product we have to first compute the product modulo small primes and then combine all these products to get the right answer using iterated sum. Therefore, many basic facts of the modulo arithmetic need to be proven in advance, as well as some basic facts of the iterated sum.

Once this is done we need to obtain the same fact modulo p (Lemma 48). At this point it is easier to go through the regular product, where the basic facts of modular arithmetic are easier to prove. Therefore it is important to show that TC^0-Frege can prove

$$PROD[z_1, z_2] = z_1 \cdot z_2$$

(Lemma 47). In our application z_1 and z_2 will themselves be iterated products.

To show this fact we use the Chinese Remainder Theorem. We first prove the equality modulo small primes. This is relatively easy, since the sizes of these primes are sufficiently small ($O(\log N)$), and we can basically check all possible combinations. Once this is done, we apply the Chinese Remainder theorem to obtain the equality modulo the product of the primes, and since this product is big enough, we obtain the desired result.

Our TC^0-Frege proof of the Chinese Remainder Theorem, is different than the standard textbook one. The main fact that we need to show is that if for every j, $[R]_{q_j} = [S]_{q_j}$, then there are TC^0-Frege proofs of $[R]_Q = [S]_Q$ ($Q = q_1 \cdot \ldots \cdot q_t$). The usual proofs use some basic facts of division of primes, that would be hard to implement here. Instead we prove by induction on $i < t$ that $[R]_{Q_i} = [S]_{Q_i}$, where $Q_i = \prod_{j=1}^{i} q_j$. This method allows us to work with numbers smaller than the q_i's and again since these numbers are sufficiently small, we can verify all possibilities.

6 Discussion and open problems

We have shown that TC^0-Frege does not have feasible interpolation, assuming that factoring of Blum integers is not efficiently computable. This implies (under the same assumptions) that TC^0-Frege as well as any system that can polynomially-simulate TC^0-Frege is not automatizable. It is interesting to note that our proof and even the definition of the Diffie-Hellman formula itself is nonuniform, essentially due to the nonuniform nature of the iterated product formulas that we use. It would be interesting to know to what extent our result holds in the uniform TC^0 proof setting.

A recent paper [BDGMP] extends our results to prove that bounded-depth Frege doesn't have feasible interpolation assuming factoring Blum integers is sufficiently hard (actually their assumptions are stronger than ours). As a consequence bounded-depth Frege is not automatizable under the same hardness assumptions.

An important question that is still open is whether resolution, or some restricted forms of it, is automatizable. A positive answer to this question would have important applied consequences.

7 Formal proof of main lemma

The goal of this section is to prove Lemma 4. As mentioned earlier, we will build up to the proof of this lemma, by showing that basic facts concerning arithmetic, multiplication, iterated multiplication and modulus computations can be efficiently carried out in our proof system. Before we begin the formal presentation, we would like to note that we will be giving is a precise description of a sequence of lemmas that are sufficient in order to carry out a full, formal proof of Lemma 4. However, since there are many lemmas and many of them have obvious proofs, we will describe at a meta-level what is required in order to formalize the argument in TC^0-Frege, rather than give an excessively formal TC^0-Frege proof of the lemma.

In what follows, x, y and z will be numbers. Each one of them will denote a vector of n variables or formulas (representing the number), where $n \leq N$ and x_i (respectively y_i, z_i) denotes the i^{th} variable of x (representing the i^{th} bit of the number x). When we need to talk about more than three numbers, we will write $z_1, ..., z_m$ to represent a sequence of m n-bit numbers, (where $m, n \leq N$), and now $z_{i,j}$ is the j^{th} variable of z_i (representing the j^{th} bit in the i^{th} number).

Recall that whenever we say below "there are TC^0-Frege proofs" we actually mean to say "there are polynomial size TC^0-Frege proofs". Some trivial properties like $x = y \wedge y = z \rightarrow x = z$ are not stated here.

7.1 Some basic properties of addition, subtraction and multiplication

Lemma 5 *For every x, y, there are TC^0-Frege proofs of $x + y = y + x$.*

Proof of Lemma 5 Immediate from the fact that the addition formula was defined in a symmetric way. □

Lemma 6 *For every x, y, z, there are TC^0-Frege proofs of $x + (y + z) = (x + y) + z$.*

Proof of Lemma 6 By the definition of the addition formula, the i^{th} bit of $((x + y) + z)$ is equal to $\oplus_1(\oplus_1(x_i, y_i, C_i(x, y)), z_i, C_i((x + y), z))$, where $C_i(x, y)$ is the carry bit going into the i^{th} position, when we add x and y, and $C_i((x + y), z)$ is similarly defined to be the carry bit going into the i^{th} position when we add $(x + y)$ and z.

Using basic properties of $\oplus_1$ and the above definitions, there is a simple TC^0-Frege proof that if $\oplus_1(C_i(x, y), C_i((x + y), z)) = \oplus_1(C_i(y, z), C_i(x, (y + z)))$, then it follows that $((x + y) + z) = (x + (y + z))$. Thus it is left to show that for all $i \leq n + 2$,

$$\oplus_1(C_i(x, y), C_i((x + y), z)) = \oplus_1(C_i(y, z), C_i(x, (y + z))).$$

We will show how to prove the stronger equality

$$C_i(x,y) + C_i((x+y),z) = C_i(y,z) + C_i(x,(y+z)).$$

(It can be verified that this is the strongest equality possible for the 4 quantities $C_i(x,y), C_i((x+y),z), C_i(y,z), C_i(x,(y+z))$. That is, all 6 assignments for $C_i(x,y), C_i((x+y),z), C_i(y,z), C_i(x,(y+z))$ that satisfy the above equality are actually possible.)

We will prove this by induction on i. For $i = 1$, the carry bits going into the first position are zero, so the above identity holds trivially. To prove the above equality for i, we assume that it holds for $i-1$. We will prove the equality by considering many cases, where a particular case will assume a fixed value to each of the following seven quantities: $x_{i-1}, y_{i-1}, z_{i-1}, C_{i-1}(x,y), C_{i-1}(y,z), C_{i-1}((x+y),z)$, $C_{i-1}(x,(y+z))$, subject to the condition that $C_{i-1}(x,y) + C_{i-1}((x+y),z) = C_{i-1}(y,z) + C_{i-1}(x,(y+z))$. It is easy to check that the number of cases is 48 since there are 2 choices for x_{i-1}; 2 choices for y_{i-1}; 2 choices for z_{i-1}; and 6 choices in total for $C_{i-1}(x,y)$, $C_{i-1}((x+y),z)$, $C_{i-1}(y,z)$ and $C_{i-1}(x,(y+z))$.

Each case will proceed in the same way. We will first show how to compute $C_i(x,y), C_i(y,z), C_i((x+y),z)$ and $C_i(x,(y+z))$ using the above seven values. Then we simply verify that in all 48 cases where the inductive hypothesis holds, the equality is true.

First, we will show that

$$C_i(x,y) = 1 \leftrightarrow x_{i-1} + y_{i-1} + C_{i-1}(x,y) \geq 2$$

This requires a proof along the following lines. If $x_{i-1} = y_{i-1} = 1$, then the left-hand side of the above statement is true since position $i-1$ generates a carry, and also the right-hand side of the statement is also true. Similarly, if $x_{i-1} = y_{i-1} = 0$, then both sides of the above statement are false (since position $i-1$ absorbs a carry). The last case is when $x_{i-1} = 1$ and $y_{i-1} = 0$ (or vice-versa). In this case, position $i-1$ propagates a carry, so the i^{th} carry bit is 1 if and only if there exists a $j < i-1$ such that the j^{th} position generates a carry, and all positions between j and $i-1$ propagate carries – but this is exactly the definition of $C_{i-1}(x,y)$. Thus, we have in this last case that both sides of the statement are true if and only if $C_{i-1}(x,y)$ is true.

Using the above fact, and also that $(x+y)_i = x_i \oplus y_i \oplus C_i(x,y)$, it follows that $C_i((x+y),z) = 1$ if and only if $z_{i-1} + (x_{i-1} \oplus y_{i-1} \oplus C_{i-1}(x,y)) + C_{i-1}((x+y),z) \geq 2$. Identical arguments show that $C_i(y,z) = 1$ and $C_i(x,(y+z)) = 1$ can also be computed as simple formulas of the seven pieces of information. □

Lemma 7 *For every x, y, there are TC^0-Frege proofs of $(x+y) - y = x$.*

Proof of Lemma 7 $(x+y) - y$ is computed by taking the first N bits of $(x+y) + \overline{y} + 1$. Note that by the definition of the addition formula it follows easily that all bits of $(y + \overline{y})$ are 1, and hence that $((y+\overline{y}) + 1) = 2^N$. Thus,

$$(x+y) + \overline{y} + 1 = x + ((y+\overline{y}) + 1) = x + 2^N,$$

and hence the first N bits of this number are the same as the first N bits of x. □

Lemma 8 *For every x, y, there are TC^0-Frege proofs of $x \geq y \rightarrow (x-y)+y=x$.*

Proof of Lemma 8 $(x-y)$ is computed by taking the first N bits of $x+\overline{y}+1$. By the definition of the addition formula, and since $x \geq y$, it can be proved that the $(N+1)^{th}$ bit of $x+\overline{y}+1$ is 1, and hence that

$$(x-y)+2^N = x+\overline{y}+1.$$

Therefore, as in Lemma 7,

$$(x-y)+y+2^N = x+\overline{y}+y+1 = x+2^N.$$

In particular, the first N bits of $(x-y)+y+2^N$ are the same as those of $x+2^N$. Thus, $(x-y)+y=x$. □

Lemma 9 *For every x, y, z, there are TC^0-Frege proofs of $x+z=y+z \rightarrow x=y$.*

Proof of Lemma 9 Follows immediately from Lemma 7 and Lemma 6 as follows: $x=(x+z)-z=(y+z)-z=y$. □

Lemma 10 *For every z, there are TC^0-Frege proofs of*

$$z = SUM_i[2^{i-1}z_i].$$

Proof of Lemma 10 We need to show that for every j,

$$z_j = [SUM_i[2^{i-1}z_i]]_j.$$

This is shown by a rather tedious but straightforward proof following the definition of the formula SUM for iterated addition. Namely, we show first that

$$H = z_n..z_{n-l+1}0..0z_{n-2l}..z_{n-3l+1}0..0.....z_{2l}..z_{l+1}0..0$$

and similarly that

$$L = 0..0z_{n-l}..z_{n-2l+1}0..0z_{n-3l}..z_{n-4l+1}.....0..0z_l..z_1$$

Secondly, we show that $[H+L]_j = z_j$, using the definition of $+$. This second step is not difficult because all carry bits are zero. □

Lemma 11 *For every $z_1, ..., z_m$, and every fixed permutation α, there are TC^0-Frege proofs of*

$$SUM[z_1, ..., z_m] = SUM[z_{\alpha(1)}, ..., z_{\alpha(m)}].$$

(That is, the iterated sum is symmetric.)

Proof of Lemma 11 Immediate from the fact that the formula SUM was defined in a symmetric way. □

Lemma 12 *For every z, there are TC^0-Frege proofs of*

$$SUM[z] = z.$$

Proof of Lemma 12 By definition of the iterated addition formula SUM, it is straightforward to prove that

$$H = z_n..z_{n-l+1}0..0z_{n-2l}..z_{n-3l+1}0..0.....z_{2l}..z_{l+1}0..0$$

and similarly that

$$L = 0..0z_{n-l}..z_{n-2l+1}0..0z_{n-3l}..z_{n-4l+1}.....0..0z_l..z_1$$

Then it is also straightforward to show, using the definition of the formula for $+$, that $[H+L]_j = z_j$ for every j. (Again, all carry bits are zero.) $\square$

Lemma 13 *For every $z_1, ..., z_m$, there are TC^0-Frege proofs of*

$$SUM[z_1, ..., z_m] = z_1 + SUM[z_2, ..., z_m].$$

Proof of Lemma 13 Recall that $SUM[z_1, ..., z_m]$ is computed by adding two numbers H, L. Recall that L is computed by first computing the numbers $L_k = \sum_{i=1}^m L_{i,k}$, where $L_{i,k}$ is the low-order half of the k^{th} block of z_i. The first equality follows from Lemma 13, and Similarly, $SUM[z_2, ..., z_m]$ is computed by $H' + L'$, where L' is computed by first computing the numbers $L'_k = \sum_{i=2}^m L_{i,k}$. We can also write $z_1 = H'' + L''$, where $H'' = \sum_{k=1}^r H_{1,k} \cdot 2^l \cdot 2^{(k-1)2l}$, and $L'' = \sum_{k=1}^r L_{1,k} \cdot 2^{(k-1)2l}$.

In both L_k, L'_k the sum is computed using poly-size threshold gates, e.g. by using the unary representation of each $L_{i,k}$. It is therefore straightforward to prove for each k, $L_k = L'_k + L_{1,k}$, (e.g. by trying all the possibilities for $L'_k, L_{1,k}$, and proving the formula separately for each possibility).

Now consider the formula $L' + L''$. Since in this addition there is no carry flow from one block to the next one, and since the bits of L, L', L'' in each block are just the bits of $L_k, L'_k, L_{1,k}$ (respectively), we can conclude that $L = L' + L''$. Since in a similar way we can prove that $H = H' + H''$, we are now able to conclude

$$SUM[z_1, ..., z_m] = H + L =$$

$$(H'' + L'') + (H' + L') = z_1 + SUM[z_2, ..., z_m].$$

$\square$

Lemma 14 *For every $z_1, ..., z_m$, there are TC^0-Frege proofs of*

$$SUM[z_1 + z_2, z_3, ..., z_m] = SUM[z_1, z_2, ..., z_m].$$

Proof of Lemma 14 Can be proved easily from Lemma 13, and Lemma 6 as follows:

$$SUM[z_1, ..., z_m] = z_1 + SUM[z_2, ..., z_m]$$

$$= z_1 + (z_2 + SUM[z_3, ..., z_m]) = (z_1 + z_2) + SUM[z_3, ..., z_m]$$

$$= SUM[z_1 + z_2, z_3, ..., z_m].$$

$\square$

Lemma 15 *For every $z_1, ..., z_m$, and every $1 \leq k \leq m$, there are TC^0-Frege proofs of*

$$SUM[z_1, ..., z_{k-1}, SUM[z_k, ..., z_m]] = SUM[z_1, ..., z_m].$$

Proof of Lemma 15 By Lemma 13, Lemma 14 and Lemma 11, we have

$$SUM[z_1, ..., z_{k-1}, SUM[z_k, ..., z_m]] =$$

$$SUM[z_1, ..., z_{k-1}, z_k + SUM[z_{k+1}, ..., z_m]] =$$

$$SUM[z_1, ..., z_{k-1}, z_k, SUM[z_{k+1}, ..., z_m]].$$

The proof now follows by repeating the same argument $m - k$ times, where Lemma 12 is used for the base case. □

Lemma 16 *For every x, y, there are TC^0-Frege proofs of*

$$x \cdot y = y \cdot x.$$

Proof of Lemma 16 Immediate from the fact that the product formula was defined in a symmetric way. □

Lemma 17 *For every x, y, z, where x is a power of 2, there are TC^0-Frege proofs of*

$$x \cdot (y + z) = x \cdot y + x \cdot z.$$

Proof of Lemma 17 It is straightforward to prove that $2^i \cdot y$ where y is any sequence of bits, consists of adding to the end of y, i 0's. The lemma easily follows. □

Lemma 18 *For every $z_1, ..., z_m$, and every x where x is a power of 2, there are TC^0-Frege proofs of*

$$x \cdot SUM[z_1, ..., z_m] = SUM[x \cdot z_1, ..., x \cdot z_m].$$

Proof of Lemma 18 The proof of this lemma is like the proof of Lemma 21, but using Lemma 17 instead of 20. □

Lemma 19 *For every x, y, z, where x, y are powers of 2 there are TC^0-Frege proofs of*

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z.$$

Proof of Lemma 19 Same as the proof of Lemma 17. □

The following three lemmas are generalizations of the previous three lemmas.

Lemma 20 *For every x, y, z there are TC^0-Frege proofs of*

$$x \cdot (y + z) = x \cdot y + x \cdot z.$$

Proof of Lemma 20 By definition of the product formula,

$$x \cdot (y+z) = SUM_{i,j}[2^{i+j-2} x_i (y+z)_j].$$

Similarly,

$$x \cdot y + x \cdot z = SUM_{i,j}[2^{i+j-2} x_i y_j] + SUM_{i,j}[2^{i+j-2} x_i z_j].$$

By iterative application of Lemma 15 (using also Lemma 11),

$$SUM_{i,j}[2^{i+j-2} x_i (y+z)_j] = SUM_i[SUM_j[2^{i+j-2} x_i (y+z)_j]].$$

Similarly (using also Lemma 13, and Lemma 14),

$$SUM_{i,j}[2^{i+j-2} x_i y_j] + SUM_{i,j}[2^{i+j-2} x_i z_j] =$$

$$SUM_i[SUM_j[2^{i+j-2} x_i y_j + 2^{i+j-2} x_i z_j]],$$

and in the same way (using the same lemmas)

$$SUM_i[SUM_j[2^{i+j-2} x_i y_j + 2^{i+j-2} x_i z_j]] =$$

$$SUM_i[SUM_j[2^{i+j-2} x_i y_j] + SUM_j[2^{i+j-2} x_i z_j]].$$

Thus, we have to prove

$$SUM_i[SUM_j[2^{i+j-2} x_i (y+z)_j]] = SUM_i[SUM_j[2^{i+j-2} x_i y_j] + SUM_j[2^{i+j-2} x_i z_j]].$$

We will prove this by proving that for every i,

$$SUM_j[2^{i+j-2} x_i (y+z)_j] = SUM_j[2^{i+j-2} x_i y_j] + SUM_j[2^{i+j-2} x_i z_j].$$

If $x_i = 0$ this is trivial. Otherwise, $x_i = 1$ and using Lemma 19, Lemma 18, and Lemma 10 we have

$$SUM_j[2^{i+j-2} x_i (y+z)_j] = 2^{i-1} SUM_j[2^{j-1}(y+z)_j] = 2^{i-1} \cdot (y+z).$$

In the same way, (using also Lemma 17),

$$SUM_j[2^{i+j-2} x_i y_j] + SUM_j[2^{i+j-2} x_i z_j] = 2^{i-1} SUM_j[2^{j-1} y_j] + 2^{i-1} SUM_j[2^{j-1} z_j] =$$

$$2^{i-1} \cdot y + 2^{i-1} \cdot z = 2^{i-1} \cdot (y+z).$$

$\square$

Lemma 21 *For every $z_1, ..., z_m$, and every x, there are TC^0-Frege proofs of*

$$x \cdot SUM[z_1, ..., z_m] = SUM[x \cdot z_1, ..., x \cdot z_m].$$

Proof of Lemma 21 We will show that for every i,

$$x \cdot SUM[z_1, ..., z_i] + SUM[x \cdot z_{i+1}, ..., x \cdot z_m] = x \cdot SUM[z_1, ..., z_{i+1}] + SUM[x \cdot z_{i+2}, ..., x \cdot z_m].$$

The lemma then follows by the combination of all these equalities. The case $i = 0$ is proven as follows:

$$\begin{aligned} SUM[x \cdot z_1, ..., x \cdot z_m] &= x \cdot z_1 + SUM[x \cdot z_2, ..., x \cdot z_m] \\ &= x \cdot SUM[z_1] + SUM[x \cdot z_2, ..., x \cdot z_m] \end{aligned}$$

The first equality follows by applying Lemma 13, and the second equality by Lemma 12.

For the general step:

$$\begin{aligned} x \cdot SUM[z_1, ..., z_i] + SUM[x \cdot z_{i+1}, ..., x \cdot z_m] &= x \cdot SUM[z_1, ..., z_i] + x \cdot z_{i+1} + SUM[x \cdot z_{i+2}, ..., x \cdot z_m] \\ &= x \cdot (SUM[z_1, ..., z_i] + z_{i+1}) + SUM[x \cdot z_{i+2}, ..., x \cdot z_m] \\ &= x \cdot SUM[z_1, ..., z_{i+1}] + SUM[x \cdot z_{i+2}, ..., x \cdot z_m]. \end{aligned}$$

The first equality follows from Lemmas 13 and the second equality follows from Lemma 20; the third equality follows from Lemma 13. □

Lemma 22 *For every x, y, z, there are TC^0-Frege proofs of*

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z.$$

Proof of Lemma 22 We will show that $x \cdot (y \cdot z)$ is equal to $SUM_{i,j,k}[2^{i+j+k-3} x_i y_j z_k]$. The same will be true for $(x \cdot y) \cdot z$, and the lemma follows.

By the definition of the product,

$$y \cdot z = SUM_{j,k}[2^{j+k-2} y_j z_k].$$

Hence, by Lemma 10, and two applications of Lemma 21 (and using freely Lemma 16),

$$x \cdot (y \cdot z) = SUM_i[2^{i-1} x_i] \cdot SUM_{j,k}[2^{j+k-2} y_j z_k] =$$

$$SUM_i[(2^{i-1} x_i) \cdot SUM_{j,k}[2^{j+k-2} y_j z_k]] =$$

$$SUM_i[SUM_{j,k}[(2^{i-1} x_i) \cdot 2^{j+k-2} y_j z_k]].$$

Since it can be easily verified that $(2^{i-1} x_i) \cdot 2^{j+k-2} y_j z_k = 2^{i+j+k-3} x_i y_j z_k$, the above is equal to

$$SUM_i[SUM_{j,k}[2^{i+j+k-3} x_i y_j z_k]],$$

and by an iterative application of Lemma 15 (using also Lemma 11) the above is equal to

$$SUM_{i,j,k}[2^{i+j+k-3} x_i y_j z_k].$$

□

7.2 Some basic properties of less-than

Lemma 23 *For every x, y, there are TC^0-Frege proofs of $x > y \vee y > x \vee x = y$, and also of $x > 0 \vee x = 0$.*

Proof of Lemma 23 Either there is a bit i such that i is the most significant bit where x and y differ, or not. If all bits are equal, then $x = y$. But if there is i such that it is the most significant bit where they differ, then if $x_i = 1$ and $y_i = 0$, then $x > y$, and if $x_i = 0$ and $y_i = 1$, then $y > x$. □

Lemma 24 *For every x, y, there are TC^0-Frege proofs of $x > y \rightarrow (x - y) > 0$.*

Proof of Lemma 24 By lemma 23, $(x - y) = 0 \vee (x - y) > 0$. Suppose for a contradiction that $x - y = 0$. Then $x = (x - y) + y = y$, and we get $x > x$ (which is easily proved to be false). So, $x - y > 0$. □

Lemma 25 *For every x, y, z there are TC^0-Frege proofs of $x > y \ \wedge \ y \geq z \rightarrow x > z$; and also $x \geq y \ \wedge y > z \rightarrow x > z$.*

Proof of Lemma 25 If $y = z$, then the proof of the first statement is obvious. Otherwise, suppose that i is the most significant bit where $x_i \neq y_i$ and that $x_i = 1$ and $y_i = 0$. Similarly, suppose that j is the most significant bit where $y_j \neq z_j$ and $y_j = 1$ and $z_j = 0$. If $i \geq j$, then it is easy to show that i is the most significant bit where $x_i \neq z_i$, and $x_i = 1$ and $z_i = 0$ and thus $x > z$. Similarly, if $j > i$, then j is the most significant bit where $x_j \neq z_j$ and $x_j = 1$ and $z_j = 0$ and thus $x > z$. Similar reasoning also implies the second statement in the lemma. □

Lemma 26 *For every x, z, there are TC^0-Frege proofs of $x+z \geq x$; and also $z > 0 \rightarrow x+z > x$.*

Proof of Lemma 26 If $z = 0$, then it is clear that $x + z = x$. For $z > 0$, we will show inductively for decreasing k that

$$x + SUM_{i \geq k}[2^{i-1} z_i] > x.$$

Then when $k = 1$, we have $x + SUM_i[2^{i-1} z_i] = x + z > x$, by Lemma 10.

Assuming that $z > 0$, let $z_{i'}$ be the most significant bit such that $z_{i'} = 1$. The base case of the induction will be to show that $x + SUM_{i \geq i'}[2^{i-1} z_i] > x$. Because $z_i = 0$ for all $i > i'$, and applying Lemma 12, it suffices to show that $x + z' > x$, where $z' = 2^{i'-1} z_{i'}$. There are two cases. If $x_{i'} = 0$, then $x + z'$ is equal to $x_n x_{n-1} \ldots x_{i'+1} 1 x_{i'-1} \ldots x_1$, and so clearly $x + z' > x$. The other case is when $x_{i'} = 1$. Let j be the most significant bit position greater than i' such that $x_j = 0$. One clearly exists because $x_{n+1} = 0$. Then $(x + z')_j = 1$, and $x_j = 0$ and all higher bits are equal, and thus $x + z' > x$ as desired.

For the inductive step, we assume that $x + SUM_{i \geq k}[2^{i-1} z_i] > x$, and want to show that $x + SUM_{i \geq k-1}[2^{i-1} z_i] > x$. Using the same argument as in the base case, one can prove that (a) $x + SUM_{i \geq k}[2^{i-1} z_i] + 2^{k-2} z_{k-1} \geq x + SUM_{i \geq k}[2^{i-1} z_i]$. By the inductive hypothesis, (b) $x + SUM_{i \geq k}[2^{i-1} z_i] > x$. Applying Lemma 25 to (a) and (b), we obtain $x + SUM_{i \geq k}[2^{i-1} z_i] + 2^{k-2} z_{k-1} > x$. By Lemma 13, this implies $x + SUM_{i \geq k-1}[2^{i-1} z_i] > x$, as desired. □

Lemma 27 *For every x, y, z there are TC^0-Frege proofs of $x > y \to x + z > y$.*

Proof of Lemma 27 If $z = 0$, then $x + z = x > y$. Otherwise, $z > 0 \to x + z > x$ by Lemma 26. Then by Lemma 25, $x + z > y$ as desired. □

Lemma 28 *For every x, y, z there are TC^0-Frege proofs of $x > y \to x + z > y + z$.*

Proof of Lemma 28

$$\begin{aligned} x + z &= (x - y) + y + z \\ &> y + z \end{aligned}$$

The first equality follows from Lemma 8, and the second from Lemma 26, and the fact that $x > y \to x - y > 0$. □

Lemma 29 *For every x, y, there are TC^0-Frege proofs of $y > 0 \to x \leq x \cdot y$*

Proof of Lemma 29 $x \cdot y = SUM_{i,j}[2^{i+j-2}x_i y_j]$ by definition. Also, since $y > 0$ there is a bit of y that is 1, and suppose that it is y_l. Then

$$\begin{aligned} x \cdot y &= SUM_{i,j}[2^{i+j-2}x_i y_j] \\ &= SUM_i[2^{i+l-2}x_i y_l] + SUM_{i,j,j \neq l}[2^{i+j-2}x_i y_j] \\ &\geq SUM_i[2^{i+l-2}x_i y_l] \\ &= 2^{l-1}SUM_i[2^{i-1}x_i] \\ &= 2^{l-1} \cdot x \geq x \end{aligned}$$

□

7.3 Some basic properties of modular arithmetic

Recall that the formulas for $[z]_p$ and $div_p(z)$ take as inputs not only the variables p and z, but also variables k_i, r_i (for every $1 \leq i \leq n$), and variables $p_1, \ldots, p_n$. The formulas give the right output if $2^i = p \cdot k_i + r_i$, $r_i < p$ and $p_i = i \cdot p$ (for all $1 \leq i \leq n$). So the following theorems will all have the hypothesis that the values for the variables k_i, r_i and p_i are correct, and that there are short TC^0-Frege proofs for $2^i = p \cdot k_i + r_i$, $r_i < p$ and $p_i = i \cdot p$. We will state this hypothesis for the first lemma, and omit it afterwards for simplicity. For simplicity, we will also use the notations $k_0 = 0$ and $r_0 = 1$, thus $2^0 = p \cdot k_0 + r_0$.

The lemmas will be used with either $p = P$, where P is the number used for the DH formula, or with $p = q$, where q is some fixed hardwired value (e.g., $q = q_j$ or $q = Q$, where q_j is one of the primes used for the iterated product formula, and Q is the product of all these primes). If $p = q$ for some hardwired q then k_i, r_i, and $p_1, \ldots p_n$ can also be hardwired. Hence, there values are correct and it is straightforward to check (i.e. to prove) that the non-variable formulas $2^i = p \cdot k_i + r_i$, $r_i < p$ and $p_i = i \cdot p$ are all correct. If $p = P$ then k_i, r_i, and $p_1, \ldots, p_n$ are inputs for the DH formula itself, and the requirements $2^i = p \cdot k_i + r_i$, $r_i < p$ and $p_i = i \cdot p$ are part of the requirements in the DH formula.

Lemma 30 *Let z and p be n-bit numbers. Then there are TC^0-Frege proofs of*

$$2^i = k_i \cdot p + r_i, 0 \leq r_i < p, p_i = i \cdot p \text{ (for all } 0 \leq i \leq n) \longrightarrow z = SUM_i[(r_{i-1} + p \cdot k_{i-1})z_i].$$

Proof of Lemma 30 From Lemma 10, and if $2^i = r_i + p \cdot k_i$,

$$z = SUM_i[2^{i-1}z_i] = SUM_i[(r_{i-1} + p \cdot k_{i-1})z_i]$$

□

Lemma 31 *For every z and p, there are TC^0-Frege proofs of*

$$z = [z]_p + div_p(z) \cdot p.$$

Also, the following uniqueness property has a TC^0-Frege proof: If $z = x + y \cdot p$ where $0 \leq x < p$, then $x = [z]_p$ and $y = div_p(z)$.

Proof of Lemma 31 From the previous lemma, we can express z as $SUM_i[(r_{i-1} + p \cdot k_{i-1})z_i]$. Let l be the same as in the definition of the modulo formulas. Then

$$\begin{aligned}
[z]_p + p \cdot div_p(z) &= (SUM_i[r_{i-1}z_i] - p \cdot l) + p \cdot (SUM_i[k_{i-1}z_i] + l) \\
&= (SUM_i[r_{i-1}z_i] - p \cdot l) + (p \cdot SUM_i[k_{i-1}z_i] + p \cdot l) \\
&= ((SUM_i[r_{i-1}z_i] - p \cdot l) + p \cdot l) + p \cdot SUM_i[k_{i-1}z_i] \\
&= SUM_i[r_{i-1}z_i] + p \cdot SUM_i[k_{i-1}z_i] \\
&= SUM_i[r_{i-1}z_i] + SUM_i[p \cdot k_{i-1}z_i] \\
&= SUM[SUM_i[r_{i-1}z_i], SUM_i[p \cdot k_{i-1}z_i]] \\
&= SUM[r_0z_1, .., r_{n-1}z_n, p \cdot k_0z_1, .., p \cdot k_{n-1}z_n] \\
&= SUM_i[(r_{i-1} + p \cdot k_{i-1})z_i] \\
&= z
\end{aligned}$$

The first equality follows from the definitions of the formulas $[z]_p$ and $div_p(z)$. The remaining equalities follow from the following Lemmas: 20, 6, 8, 21, 13, 15, 14 and 30.

Let us now prove the uniqueness part. Suppose $z = [z]_p + div_p(z) \cdot p = x + y \cdot p$, where $0 \leq x, [z]_p < p$. If $div_p(z) = y$, then we are done. But if $div_p(z) > y$, then by the claim bellow $x \geq p$, which is a contradiction. (And a similar argument holds when $div_p(z) < y$.)

Claim 32 *If $x + y \cdot p = u + v \cdot p$ and $v > y$, then $x \geq p$.*

Proof of the claim Since $v > y$, by Lemma 8, and Lemma 24 $y + (v - y) = v$, and $v - y > 0$. Then $x + y \cdot p = u + (y + (v - y)) \cdot p$, and by Lemma 20, $x + y \cdot p = u + y \cdot p + (v - y) \cdot p$. By Lemma 9 we get that $x = u + (v - y) \cdot p$. Therefore by Lemmas 29 and 26,

$$p \leq (v - y) \cdot p \leq u + (v - y) \cdot p = x$$

and by Lemma 25 we get $p \leq x$. □

Lemma 33 *For every z, k and p, there are TC^0-Frege proofs of*

$$[z]_p = [z + k \cdot p]_p.$$

Proof of Lemma 33 Let $x = z + k \cdot p$. Then $x = [x]_p + div_p(x) \cdot p$ and $z = [z]_p + div_p(z) \cdot p$. (by Lemma 31).

So, $z + k \cdot p = x = [x]_p + div_p(x) \cdot p$. Therefore, $[z]_p + div_p(z) \cdot p + k \cdot p = [x]_p + div_p(x) \cdot p$ By the uniqueness part of Lemma 31 applied to x, $[z]_p = [x]_p$. □

Lemma 34 *For every x, y, z and p, there are TC^0-Frege proofs of*

$$[x + y]_p = [[x]_p + [y]_p]_p,$$

and also of

$$[x + y]_p = [[x]_p + y]_p,$$

and

$$[x + y + z]_p = [[x]_p + [y]_p + z]_p.$$

Proof of Lemma 34 By Lemma 31, $x = [x]_p + div_p(x) \cdot p$ and $y = [y]_p + div_p(y) \cdot p$. Hence,

$$[x + y]_p = [[x]_p + [y]_p + (div_p(x) + div_p(y)) \cdot p]_p,$$

and by Lemma 33,

$$[x + y]_p = [[x]_p + [y]_p]_p.$$

A similar argument shows that $[x + y]_p = [[x]_p + y]_p$, and $[x + y + z]_p = [[x]_p + [y]_p + z]_p$. □

Lemma 35 *For every $z_1, ..., z_m$ and p, there are TC^0-Frege proofs of*

$$SUM[z_1, ..., z_m]_p = [[z_1]_p + SUM[z_2, ..., z_m]_p]_p.$$

Proof of Lemma 35 The lemma follows easily from Lemma 13 and Lemma 34. □

Lemma 36 *For every x, y and p, there are TC^0-Frege proofs of*

$$[x]_p = [y]_p \longrightarrow [x + z]_p = [y + z]_p.$$

Proof of Lemma 36

$$[x + z]_p = [[x]_p + [z]_p]_p = [[y]_p + [z]_p]_p = [y + z]_p$$

The first equality follows from Lemma 34; the next equality follows from the assumption that $[x]_p = [y]_p$ and the third equality follows from Lemma 34. □

Lemma 37 *For every x, y, z and p, there are TC^0-Frege proofs of*

$$[x+z]_p = [y+z]_p \longrightarrow [x]_p = [y]_p.$$

Proof of Lemma 37 Assuming that $[x+z]_p = [y+z]_p$, it follows from the above Lemma 36 that $[x+z+(p-[z]_p)]_p = [y+z+(p-[z]_p)]_p$. The left side of the equation is equal to:

$$\begin{aligned} [x+z+(p-[z]_p)]_p &= [[x]_p + [z]_p + (p-[z]_p)]_p \\ &= [[x]_p + p]_p \\ &= [x]_p \end{aligned}$$

The first equality follows from Lemma 34; the second equality follows from Lemmas 5, 6 and 7; the third equality follows from Lemma 33. Similarly, it can be shown that $[y+z+(p-[z]_p)]_p = [y]_p$ and thus the lemma follows. □

Lemma 38 *For every x, y and p, there are TC^0-Frege proofs of*

$$[x \cdot y]_p = [x \cdot [y]_p]_p.$$

Proof of Lemma 38

$$\begin{aligned} [x \cdot y]_p &= [x \cdot ([y]_p + div_p(y) \cdot p)]_p \\ &= [x \cdot [y]_p + x \cdot div_p(y) \cdot p]_p \\ &= [x \cdot [y]_p]_p, \end{aligned}$$

where the last equality follows from Lemma 33. □

Lemma 39 *Let A, B, C be fixed numbers such that $A = BC$. Then for every z, there are TC^0-Frege proofs of*

$$[z]_B = [[z]_A]_B.$$

This lemma will be used in situations where $A = Q$ and $B = q_i$ for some i. Recall that the numbers $Q, q_1, ...q_t$ are hardwired and also their corresponding k_i, r_i and the variables for the $j \cdot q_i$'s. Hence, we think of A, B, C as hardwired.

Proof of Lemma 39 Using Lemmas 31,33,22, we get

$$[z]_B = [[z]_A + A \cdot div_A(z)]_B = [[z]_A + B \cdot (C \cdot div_A(z))]_B = [[z]_A]_B.$$

□

7.4 Some basic properties of iterative product

Lemma 40 *For every $z_1, ..., z_m$, and every fixed permutation α, there are TC^0-Frege proofs of*

$$PROD[z_1, ..., z_m] = PROD[z_{\alpha(a)}, ..., z_{\alpha(m)}].$$

(That is, the iterated product is symmetric.)

Proof of Lemma 40 This lemma is immediate from the symmetric definition of $PROD$. $\square$

Lemma 41 *For every $z_1, ..., z_m$, and every $1 \le k \le m$, there are TC^0-Frege proofs of*

$$PROD[z_1, ..., z_m] = PROD[z_1, ..., z_{k-1}, PROD[z_k, ..., z_m]].$$

Proof of Lemma 41 Recall that we have hard-coded the numbers u_j, such that $u_j \bmod q_j = 1$ and for all $i \ne j$, $u_j \bmod q_i = 0$. For all primes q_j dividing Q, and for all m, $1 \le m \le q_j$, we can verify the following statements: $[u_j \cdot m]_{q_j} = m$, and for all $i \ne j$, $[u_j \cdot m]_{q_i} = 0$. (Note that these statements are variable-free and hence they can be easily proven by doing a formula evaluation.)

Recall that for any k, the iterated product of the numbers $z_k, ..., z_m$ is calculated as follows:

$$PROD[z_k, ..., z_m] = SUM_{j=1}^t [u_j \cdot r_j^{[k,..,m]}]_Q,$$

where $r_j^{[k,...,m]}$ is computed like r_j as defined in Section 3.4, but using r_{ij} only for i such that $k \le i \le m$.

In the same way,

$$PROD[z_1, .., z_{k-1}, PROD[z_k, ..., z_m]] =$$
$$SUM_{j=1}^t [u_j \cdot r_j^{[1,...,k-1,[k,..,m]]}]_Q,$$

where $r_j^{[1,..,k-1,[k,..,m]]}$ is calculated as before by the following steps:

1. For $i < k$, calculate $r_{i,j} = [z_i]_{q_j}$, and also calculate $r_{*,j} = PROD[z_k, ..., z_m]_{q_j}$.
2. For $i < k$ calculate $a_{i,j}$ such that $(g_j^{a_{i,j}}) \bmod q_j = r_{i,j}$, and also $a_{*,j}$ such that $(g_j^{a_{*,j}}) \bmod q_j = r_{*,j}$ by table-lookup.
3. Calculate $c'_j = SUM[a_{1,j}, ..., a_{k-1,j}, a_{*,j}]_{(q_j-1)}$.
4. Calculate $r_j^{[1,..,k-1,[k,..,m]]}$ such that $g^{c'_j} \bmod q_j = r_j^{[1,...,k-1,[k,..,m]]}$ by table-lookup.

Therefore, all we have to do is to show that

$$SUM_{j=1}^t [u_j \cdot r_j^{[1,...,m]}]_Q = SUM_{j=1}^t [u_j \cdot r_j^{[1,...,k-1,[k,..,m]]}]_Q.$$

Hence, all we need to do to prove Lemma 41 is to show the following claim:

Claim 42 *For every j, there are TC^0-Frege proofs of $r_j^{[1,...,k-1,[k,...,m]]} = r_j^{[1,..,m]}$.*

The first step is to prove the following claim:

Claim 43 *There are TC^0-Frege proofs of $PROD[z_k, .., z_m]_{q_j} = r_j^{[k,...,m]}$.*

Claim 43 is proven as follows.

$$PROD[z_k, ..., z_m]_{q_j} =$$

$$[SUM_{i=1}^{t}[u_i \cdot r_i^{[k,...,m]}]_Q]_{q_j} = SUM_{i=1}^{t}[u_i \cdot r_i^{[k,...,m]}]_{q_j} =$$

$$[[u_j \cdot r_j^{[k,...,m]}]_{q_j} + SUM_{i \neq j}[u_i \cdot r_i^{[k,..,m]}]_{q_j}]_{q_j} =$$

$$[r_j^{[k,..,m]} + 0]_{q_j} = r_j^{[k,...,m]}.$$

The second equality follows by Lemma 39; the third equality follows by Lemma 35, and Lemma 11. To prove the fourth equality, we need to use the fact that $[u_j \cdot r_j^{[k,...,m]}]_{q_j} = r_j^{[k,...,m]}$, and also for all $i \neq j$, $[u_i \cdot r_i^{[k,...,m]}]_{q_j} = 0$. These facts can be easily proved just by checking all possibilities for $r_i^{[k,...,m]}$ (proving the statement for each possibility is easy, because these statements are variable-free and hence they can be easily proven by doing a formula evaluation). In order to prove the fourth equality formally, we can show that $SUM_{i \neq j}[u_i \cdot r_i^{[k,...,m]}]_{q_j}$ equals zero by induction on the number of terms in the sum. □

We can now turn to the proof of Claim 42. The quantity $r_j^{[1,...,m]}$ is obtained by doing a table lookup to find the value equal to $g_j^{c_j} mod q_j$, where $c_j = SUM_{i=1}^{m}[a_{i,j}]_{(q_j-1)}$. Similarly, the quantity $r_j^{[1,...,k-1,[k,...,m]]}$ is obtained by doing a table lookup to find the value equal to $g_j^{c'_j} mod q_j$, where $c'_j = SUM[a_{1,j}, a_{2,j}, .., a_{k-1,j}, a_{*,j}]_{(q_j-1)}$.

Hence, it is enough to prove that $c_j = c'_j$. Using previous lemmas

$$c_j = [SUM_{i=1}^{k-1}[a_{i,j}]_{(q_j-1)} + SUM_{i=k}^{m}[a_{i,j}]_{(q_j-1)}]_{(q_j-1)}.$$

$$c'_j = [SUM_{i=1}^{k-1}[a_{i,j}]_{(q_j-1)} + a_{*,j}]_{(q_j-1)}.$$

Thus, it suffices to show that

$$SUM_{i=k}^{m}[a_{i,j}]_{(q_j-1)} = a_{*,j}.$$

Recall that $a_{*,j}$ is the value obtained by table-lookup such that $(g_j^{a_{*,j}})$ mod $q_j = r_{*,j}$, and by Claim 43, we have that $r_{*,j} = r_j^{[k,...,m]}$. Now $r_j^{[k,...,m]}$, in turn, is the value obtained by table-lookup to equal (g_j^d) mod q_j, where $d = SUM_{i=k}^{m}[a_{i,j}]_{(q_j-1)}$.

Now it is easy to verify that our table-lookup is one-to-one. That is, for every $x, y, z \leq q_j$, if g_j^x mod $q_j = z$, and g_j^y mod $q_j = z$, then $x = y$. Using this property (with $x = SUM_{i=k}^{m}[a_{i,j}]_{(q_j-1)}$, $y = a_{*,j}$ and $z = r_{*,j} = r_j^{[k,...,m]}$), it follows that

$$SUM_{i=k}^{m}[a_{i,j}]_{(q_j-1)} = a_{*,j}.$$

□

7.5 The Chinese Remainder Theorem and other properties of iterative product

The heart of our proof is a TC^0-Frege proof for the following lemma, which gives the hard direction of the Chinese Remainder Theorem (a TC^0-Frege proof for the other direction is simpler).

Lemma 44 *Let R, S be two integers, such that for every j, $[R]_{q_j} = [S]_{q_j}$. Then there are TC^0-Frege proofs of*

$$[R]_Q = [S]_Q.$$

(where $q_1, ..., q_t$ are the fixed primes used for the $PROD$ formula (i.e., the first t primes), and Q is their product.)

Proof of Lemma 44 Without loss of generality, we can assume that $0 \leq R, S \leq Q - 1$, and prove that $R = S$. Otherwise, define $R' = [R]_Q$, and $S' = [S]_Q$, and use Lemma 39 to show that for every j, $[R']_{q_j} = [S']_{q_j}$. Since $0 \leq R', S' \leq Q - 1$, we can then conclude that

$$[R]_Q = R' = S' = [S]_Q.$$

For every k, let Q_k denote $\prod_{j=1}^{k} q_j$. Note that the numbers Q_k can be hardwired, and that one can easily prove the following statements. (These statements are variable-free and hence they can be easily proven by doing a formula evaluation.)

For every i, $Q_{i+1} = Q_i \cdot q_{i+1}$.

The proof of the lemma is by induction on t (the number of q_j's). For $t = 1$, $Q = q_1$, and the lemma is trivial. Assume therefore by the induction hypothesis that

$$[R]_{Q_{t-1}} = [S]_{Q_{t-1}},$$

and hence

$$[[R]_{Q_{t-1}}]_{q_t} = [[S]_{Q_{t-1}}]_{q_t}.$$

Denote, $D_R = div_{Q_{t-1}}[R]$, and $D_S = div_{Q_{t-1}}[S]$. Then by Lemma 31,

$$R = D_R \cdot Q_{t-1} + [R]_{Q_{t-1}},$$

and

$$S = D_S \cdot Q_{t-1} + [S]_{Q_{t-1}},$$

and since we know that $[R]_{q_t} = [S]_{q_t}$, we have

$$[D_R \cdot Q_{t-1} + [R]_{Q_{t-1}}]_{q_t} = [D_S \cdot Q_{t-1} + [S]_{Q_{t-1}}]_{q_t},$$

and by $[R]_{Q_{t-1}} = [S]_{Q_{t-1}}$, and Lemma 37

$$[D_R \cdot Q_{t-1}]_{q_t} = [D_S \cdot Q_{t-1}]_{q_t}.$$

Since R, S are both lower than Q, it follows that D_R, D_S are both lower than q_t. Hence, by Claim 45 $D_R = D_S$. Therefore, we can conclude that

$$R = D_R \cdot Q_{t-1} + [R]_{Q_{t-1}} = D_S \cdot Q_{t-1} + [S]_{Q_{t-1}} = S.$$

□

Claim 45 *For every i, there are TC^0-Frege proofs of: if $d_1, d_2 < q_i$, and $[d_1 \cdot Q_{i-1}]_{q_i} = [d_2 \cdot Q_{i-1}]_{q_i}$ then $d_1 = d_2$.*

Proof Since $d_1, d_2 < q_i$, there are only $O(\log n)$ possibilities for d_1, d_2. Therefore, one can just check all the possibilities for d_1, d_2. Proving the statement for each possibility is easy, because these statements are variable-free and hence they can be easily proven by doing a formula evaluation.

Alternatively, one can define the function $f(x) = [x \cdot Q_{i-1}]_{q_i}$, in the domain $\{0, ..., q_i\}$, and prove that $f(x)$ is onto the range $\{0, ..., q_i\}$. Then, by applying the propositional pigeonhole principle, which is efficiently provable in TC^0-Frege, it follows that f is one to one. $\square$

We are now able to prove the following lemmas.

Lemma 46 *For every z, there are TC^0-Frege proofs of*

$$PROD[z] = z.$$

Proof of Lemma 46 Recall that $PROD[z]$ is calculated as follows:

$$PROD[z] = SUM_{j=1}^{t}[u_j \cdot r_j]_Q,$$

where r_j is computed by $r_j = [z]_{q_j}$.

By Claim 43, for every i, $PROD[z]_{q_i} = r_i$. We thus have for every i, $PROD[z]_{q_i} = [z]_{q_i}$. The proof of the lemma now follows by Lemma 44.

$\square$

Lemma 47 *For every z_1, z_2, there are TC^0-Frege proofs of*

$$PROD[z_1, z_2] = z_1 \cdot z_2.$$

Proof of Lemma 47 Let us prove that for every i,

$$[PROD[z_1, z_2]]_{q_i} = [z_1 \cdot z_2]_{q_i}.$$

The proof of the lemma then follows by Lemma 44. By two applications of Lemma 38 it is enough to prove for every i,

$$[PROD[z_1, z_2]]_{q_i} = [[z_1]_{q_i} \cdot [z_2]_{q_i}]_{q_i}.$$

Recall that $PROD[z_1, z_2]$ is calculated as follows:

$$PROD[z_1, z_2] = SUM_{j=1}^{t}[u_j \cdot r_j^{[1,2]}]_Q,$$

where $r_j^{[1,2]}$ is computed like r_j as defined in Section 3.4. By Claim 43, for every i,

$$PROD[z_1, z_2]_{q_i} = r_i^{[1,2]}.$$

Recall that $[z_1]_{q_i} = r_{1,i}$, and $[z_2]_{q_i} = r_{2,i}$. Therefore, all we have to prove is that for every i,

$$r_i^{[1,2]} = [r_{1,i} \cdot r_{2,i}]_{q_i}.$$

By the definitions: $r_{1,i} = (g_i^{a_{1,i}}) \bmod q_i$, and $r_{2,i} = (g_i^{a_{2,i}}) \bmod q_i$, and therefore,

$$[r_{1,i} \cdot r_{2,i}]_{q_i} = [(g_i^{a_{1,i}}) \bmod q_i \cdot (g_i^{a_{2,i}}) \bmod q_i]_{q_i}.$$

Also,

$$r_i^{[1,2]} = (g_i^{SUM[a_{1,i},a_{2,i}]_{(q_i-1)}}) \bmod q_i.$$

Therefore, one can just check all the possibilities for $a_{1,i}, a_{2,i}$.

□

Using the previous lemmas, we are now able to prove the following:

Lemma 48 *For every $z_1, ..., z_m$, every $k \leq m-1$ and every p, there are TC^0-Frege proofs of*

$$PROD[z_1, ..., z_m]_p =$$

$$PROD[z_1, ..., z_k, PROD[z_{k+1}, ..., z_m]_p]_p$$

(as before, given that $2^i = k_i \cdot p + r_i, 0 \leq r_i < p, p_i = i \cdot p$ for all i).

Proof of Lemma 48

$$PROD[z_1, .., z_k, PROD[z_{k+1}, .., z_m]_p]_p =$$

$$PROD[PROD[z_1, .., z_k], PROD[z_{k+1}, .., z_m]_p]_p =$$

$$[PROD[z_1, .., z_k] \cdot PROD[z_{k+1}, .., z_m]_p]_p =$$

$$[PROD[z_1, .., z_k] \cdot PROD[z_{k+1}, .., z_m]]_p =$$

$$PROD[PROD[z_1, .., z_k], PROD[z_{k+1}, .., z_m]]_p =$$

$$PROD[z_1, .., z_k, PROD[z_{k+1}, .., z_m]]_p =$$

$$PROD[z_1, .., z_k, z_{k+1}, .., z_m]_p.$$

The lemmas used for each equality in turn are: Lemmas 41,47, 38,47,41, and 41. □

We are now ready to prove Lemma 4:

For every $z_{1,1}, ..., z_{m,m'}$ and p, there are TC^0-Frege proofs of

$$PROD_{i,j}[z_{i,j}]_p = PROD_i[PROD_j[z_{i,j}]_p]_p,$$

(given that $2^i = k_i \cdot p + r_i, 0 \leq r_i < p, p_i = i \cdot p$ for all i).

Proof of Lemma 4 By an iterative application of the previous lemma. □

8 Acknowledgments

We are very grateful to Omer Reingold and Moni Naor for collaboration at early stages of this work, and in particular for suggesting the use of the Diffie-Hellman cryptographic scheme. We also would like to thank Uri Feige for conversations and for his insight about extending this result to bounded-depth Frege. Part of this work was done at Dagstuhl, during the Complexity of Boolean Functions workshop (1997).

References

[AB] Alon N., and Boppana R., "The Monotone Circuit Complexity of Boolean Functions", *Combinatorica*, Vol 7, No. 1, pp. 1-22, 1987.

[ABMP] Aleknovich, A., Buss, S., Moran, S., and Pitassi, T. "Minimal propositional proof length is NP-hard to linearly approximate", To appear in *Journal of Symbolic Logic*.

[BBR] E. Biham, D. Boneh and O. Reingold, "Generalized Diffie-Hellman modulo a composite is not weaker than factoring", *Theory of Cryptography Library*, Record 97-14 at: `http://theory.lcs.mit.edu/~tcryptol/homepage.html`

[BCH] Beame, P., Cook, S., and Hoover, J. " Log depth circuits for division and related problems," *SIAM J. Comput*, vol. 15 (1986), pp. 996-1003.

[BDGMP] Bonet, M., Domingo, C., Gavaldà, R., Maciel, A., Pitassi, T. "Non-automatizability of bounded-depth Frege proofs", Proceedings of the *IEEE Symposium on Complexity*, 1999.

[BPR] Bonet, M., Pitassi, T., and Raz, R. "Lower bounds for Cutting Planes proofs with small coefficients," Proceedings of the *ACM Symposium on the Theory of Computing*, 1995, pp. 575-584. Also in *Journal of Symbolic Logic*, vol 62, Numb. 3 (1997), pp. 708-728.

[B1] Buss, S. "The Undecidability of k-Provability", *Annals of Pure and Applied Logic*, vol. 53 (1991) pp. 75-102.

[B2] Buss, S. "On Godel's theorems on lengths of proofs II: Lower bounds for recognizing k symbol provability," In *Feasible Mathematics II*, P. Clote and J. Remmel, eds., Birkhaauser-Bostin, 1995, pp. 57-90.

[BC] Buss, S., Clote, P. "Cutting planes, connectivity and threshold logic," *Archive for Mathematical Logic*, vol. 35 (1996), pp. 33-62.

[CH] Cook, S., and Haken, A., "An Exponential Lower Bound for the Size of Monotone Real Circuits," accepted for JCSS.

[CSV] Chandra, A.K., Stockmeyer, L., and Vishkin, U., "Constant depth reducibility," *SIAM J. on Computing*, Vol 13(2), 1984, pp. 423-439.

[DH] Diffie, W., and Hellman, M., "New directions in cryptography," *IEEE Trans. Inform. Theory*, Vol. 22(6), 1976, pp. 644-654.

[IPU] Impagliazzo, R., Pitassi, T., Urquhart, A., "Upper and lower bounds for tree-like Cutting Planes proofs," Proceedings from the *IEEE Symposium on Logic in Computer Science*, 1994.

[K1] Krajíček J., "Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic", to appear in the *Journal of Symbolic Logic*.

[K2] Krajíček J., "Lower bounds to the size of constant-depth propositional proofs", *Journal of Symbolic Logic*, 59(1), 1994, pp. 73-86.

[K3] Krajíček J., "Discretely ordered modules as a first-order extension of the cutting planes proof system," submitted.

[KP] Krajíček J., and Pudlák , P. "Some consequences of cryptographical conjectures for S_2^1 and EF," *Logic and Computational Complexity* Ed. D. Leivant, Springer-Verlag, Lecture Notes in Computer Science, Vol. 960, (1995), pp.210-220.

[M] Maciel, Alexis, "Threshold circuits of small majority depth," Ph.D. Thesis, McGill University, February 1995.

[MP] Maciel, A., and Pitassi, T. "On $ACC^0[p^k]$-Frege proofs," Proceedings of *ACM Symposium on Theory of Computing (STOC)*, 1997, pp. 720-729.

[Mc] McCurley, K., "A key distribution system equivalent to factoring," *J. of Cryptology*, Vol 1, 1988, pp. 95-105.

[M1] Mundici, D., "Complexity of Craig's interpolation, *Fundamenta Informaticae*, 5, pp. 261-278.

[M2] Mundici, D., "A lower bound for the complexity of Craig's interpolants in sentential logic," *Archiv fur Math. Logik*, 23, pp. 27-36.

[M3] Mundici, D., "Tautologies with a unique Craig interpolant, uniform vs. non-uniform complexity, *Annals of Pure and Applied Logic*, 27, pp. 265-273.

[Pud] Pudlák , P., "Lower bounds for resolution and cutting planes proofs and monotone computations," *J. Symbolic Logic*, vol. 62, numb. 3 (1997).

[PS] Pudlák , P., and Sgall, J., "Algebraic models of computation and interpolation for algebraic proof systems," submitted.

[Pud3] Pudlák , P., "On the complexity of propositional calculus", To appear in *Logic Colloquium 97*.

[Razb1] Razborov A., "Lower Bounds for the Monotone Complexity of some Boolean Functions", *Dokl. Ak. Nauk. SSSR*, Vol 281, pp. 798-801, 1985 (in Russian). English translation in *Sov. Math. Dokl.* Vol 31, pp. 354-357, 1985.

[Razb2] Razborov, A., "Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic," *Izvestiya of the R.A.N.*, 59(1) pp.201-224, 1995.

[Sh] Shmuely, Z., "Composite Diffie-Hellman public-key generating systems are hard to break," Technical Report No. 356, Computer Science Department, Technion, Israel, 1985.