No Feasible Interpolation for TC^0-Frege Proofs

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 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. In particular, we show how to carry out the proof for the Chinese Remainder Theorem, which may be of independent interest. As a corollary, we obtain that TC^0-Frege as well as any proof system that polynomially simulates it, is not automatizable (under a hardness assumption).

1 Introduction

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. The basic idea behind the interpolation method is as follows.

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.

There is also a monotone version of the interpolation idea. Namely, $F = A_0(x,z) \wedge A_1(y,z)$ is monotone if the variables of z occur only positively in A_1 and only negatively in A_0. In this case, we are interested in finding a polynomial-size monotone circuit for an interpolant function, and 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, it was proved that a (sufficiently strong) proof system S, that admits monotone feasible interpolation, cannot have polynomial-size proofs for all tautologies. 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

*Department of LSI, Universidad Politécnica de Cataluña, Barcelona, Spain, bonet@goliat.upc.es. Research partly supported by EU HCM network console.

†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

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] such a circuit is of exponential size. Thus, exponential lower bounds follow for any propositional proof system S that admits feasible monotone interpolation.

In the non-monotone case, it was shown in [Razb2] that assuming a cryptographic assumption, if a (sufficiently strong) proof system S admits feasible interpolation then S cannot have polynomial-size proofs of all tautologies. In short, 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).

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, K2], relativized bounded arithmetic [Razb2], Hilbert's Nullstellensatz [PS], the polynomial calculus [PS], and the Lovasz-Schriver proof system [Pud3].

One of the most important questions in propositional proof complexity is to show that there is a family of propositional tautologies requiring super-polynomial-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 whether or not the interpolation method can be applied to prove lower bounds for these systems, as well as for weaker systems.

1.1 Automatizability

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 [Im]. The idea is to show that if S is automatizable (using deterministic procedure D), then S has feasible interpolation: Suppose that $A_0(x,z) \wedge A_1(y,z)$ is the interpolant statement, and let α be an assignment to z. Then we can run D on $A_0(x,\alpha) \wedge A_1(y,\alpha)$ to obtain a refutation of size s. Next, we run D on $A_0(x,\alpha)$ and return 0 if and only if D produces a refutation of $A_0(x,\alpha)$ within time $T(s)$ (where $T(s)$ is the time for D to produce a refutation for a formula that has a refutation of size s). 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.

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

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 a feasible interpolation theorem 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 have a feasible interpolation theorem.

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 is computable by polynomial-size circuits. 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]. As observed by Naor [Na], the cryptographic primitive needed here is not 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).

Fix a number P (not necessarily a prime) of length n, and let g be a generator of the group Z_P^*. Our propositional statement $DH_{P,g}$ will be

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

The common variables are two integers X, Y. The private variables for A_0 are integers a, b, and the private variables for A_1 are integers c, d.

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

$$(g^{ab} modP) = (g^a modP)^b modP = X^b modP =$$

$$(g^c modP)^b modP = g^{bc} modP = (g^b modP)^c modP =$$

$$Y^c modP = (g^d modP)^c modP = g^{cd} modP.$$

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, breaking the Diffie-Hellman cryptographic scheme is harder than factoring P ! [Sh, Mc].

It will require quite a bit of work to formalize the above argument in TC^0-Frege. In particular, it will require a TC^0-Frege proof for the Chinese-Reminder-Theorem, and TC^0-Frege proofs for the main facts of basic arithmetic.

1.4 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, $\neg$, and $\oplus$. ($Th_k(x)$ is true if and only if the number of 1's in x is at least k, and $\oplus_1(x)$ is true if and only if the number of 1's in x is odd.)

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].) The full description of our proof system is omitted in this version of the paper.

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. For simplicity, we will usually omit the words: "polynomial-size". I.e, whenever we say that a family has TC^0-Frege proofs, we actually mean to say "polynomial-size TC^0-Frege proofs".

1.5 Section description

The paper is organized as follows. In Section 2, we define the TC^0-formulas used for the proof. In Section 3, we define precisely the interpolation formulas which are based on the Diffie-Hellman cryptographic scheme. In Sections 4 and 5 we give a sketch of the proof, and we conclude with a short discussion in Section 6.

2 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 the number P, used for the Diffie-Hellman formula, is fixed, and that we also 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.

2.1 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 negation of the N bits of y, and $\overline{x}$ is the negation 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 $(s_{N+1} \wedge s_i) \vee (\neg s_{N+1} \wedge t_i)$.

2.2 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 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,

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

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

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, $L = L_r, L_{r-1}, ..., L_1$.

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.

2.3 Modulo arithmetic

Next, we describe our TC_0-formulas that compute the remainder and the largest divisor (respectively) of a number z modulo p **(where the number p is fixed, and is not an input for the formula)**. That is, suppose that $z = kp + r$, where $0 \leq r < p$. 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., $z = \sum_{i=1}^{n} 2^{i-1} z_i$. For each $i \leq n$, we will have hardwired k_i and r_i such that $2^i = p \cdot k_i + r_i$, where $0 \leq r_i < p$. We will also have hardwired the values $p, 2p, ..., np$.

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 $l \cdot p \leq s < (l+1) \cdot p$ (l can be computed just by checking all the possibilities). 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 $SUM_{i=1}^{n} [k_{i-1} \cdot z_i] + l$.

2.4 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 $u_j mod q_j = 1$ and for all $i \neq j$, $u_j mod q_i = 0$ (such a number exists by the Chinese Reminder 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 modulo arithmetic described earlier.

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

   a. Compute a_{ij} such that $(g_j^{a_{ij}}) mod q_j = r_{i,j}$. 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 $g_j^{c_j} mod q_j = r_j$. 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.

2.5 Equality, and inequality

Often we will write $x = y$, where x and y are both vectors of variables or formulas: $x = x_n, ..., x_1$, and $y = y_n, .., y_1$. 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 $\neq, <, \leq, >, \geq$.

3 The Diffie-Hellman Formula

We are now ready to formally define our propositional statement $DH_{P,g}$, for an n-bits integer P, and a generator g of the group Z^*_P. $DH_{P,g}$ will be the conjunction of A_0 and A_1. The common variables for the formulas will be X, Y, and for every $i \leq 2n$, we will also add common variables for $X^{2^i} mod P$, and for $Y^{2^i} mod P$ (that is, whenever we write $X^{2^i} mod P$ or $Y^{2^i} mod P$ we mean to these new common variables, and not to some expressions in X or Y). For every $i \leq 2n$, we will also use the number $g^{2^i} mod P$. Note that since g is fixed, we can think of the numbers $g mod P, g^2 mod P, g^4 mod P, ..., g^{2^i} mod P$ as hardwired.

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: $g^{2^i} mod P$ (respectively $X^{2^i} mod P$, $Y^{2^i} mod P$) if $e = 1$, and 1 if $e = 0$. The formula $A_0(X, Y, a, b)$ 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 $j \leq n$,

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

   (Which means $(g^{2^j})^a mod P = X^{2^j} mod P$.) 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 $g^b modP = Y$, and for $(g^{2^j})^b modP = Y^{2^j} modP$.

4.
$$PROD_{i,j}\left[g^{2^{i+j}\cdot a_i \cdot b_j}\right]_P = EVEN.$$

(Which means $g^{ab} modP$ is even.)

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

4 A TC^0-Frege refutation for $DH_{P,g}$

We want to describe a TC^0-Frege refutation for $DH_{P,g}$. As mentioned above the proof proceeds as follows.

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

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

We formulate $g^{ab} modP$ as

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

and $X^b modP$ 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 1,2,4,6 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 how to prove $g^{ab} modP = X^b modP$, which is formulated as

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

We will build up to the proof that $g^{ab} modP$ equals $X^b modP$ by first proving many lemmas concerning our basic TC^0-formulas.

In the following lemmas, x, y and z will usually 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. Also, in this version of the paper, most of the proofs are omitted.

4.1 Some basic properties of arithmetic

Lemma 1 *For every x, y, there are TC^0-Frege proofs of $x + y = y + x$.*

Lemma 2 *For every x, y, z, there are TC^0-Frege proofs of $x + (y + z) = (x + y) + z$.*

Lemma 3 *For every x, y, there are TC^0-Frege proofs of $(x + y) - y = x$.*

Lemma 4 *For every x, y, such that $x \geq y$, there are TC^0-Frege proofs of $(x - y) + y = x$.*

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

Lemma 6 *For every x, y, z, there are TC^0-Frege proofs of $x < y \rightarrow x + z < y + z$.*

Lemma 7 *For every x, y, z, there are TC^0-Frege proofs of $x \leq y \wedge y \leq z \rightarrow x \leq z$.*

Lemma 8 *For every z, there are TC^0-Frege proofs of*

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

Lemma 9 *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.)

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

$$SUM[z] = z.$$

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

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

Lemma 12 *For every $z_1, ..., z_m$, there are TC^0-Frege proofs of*

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

Lemma 13 *For every $z_1, ..., z_m$, and every $1 \le k \le m$, there are TC^0-Frege proofs of*

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

Lemma 14 *For every x, y, z, there are TC^0-Frege proofs of*

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

Lemma 15 *For every $z_1, ..., z_m$, and every x, there are TC^0-Frege proofs of*

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

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

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

4.2 Some basic properties of the modulo arithmetic

Lemma 17 *Let z be an n-bit number, and let p be a fixed number. For each $i \le n$, let $r_i = [2^{i-1}]_p$ if $z_i = 1$, and $r_i = 0$ otherwise, and let $k_i = div_p(2^{i-1})$ if $z_i = 1$ and $k_i = 0$ otherwise, where the numbers $[2^{i-1}]_p$, $div_p(2^{i-1})$ are hard-coded actual numbers. Then there are TC^0-Frege proofs of*

$$z = SUM_i[r_i + p \cdot k_i].$$

Lemma 18 *For every z and every fixed 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 $0 \le x < p$, then $x = [z]_p$ and $y = div_p(z)$.

Lemma 19 *For every z, k and every fixed p, there are TC^0-Frege proofs of*

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

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

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

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

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

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

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

Lemma 23 *For every x, y and every fixed p, there are TC^0-Frege proofs of*

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

Lemma 24 *Let A, B, C be fixed numbers such that $A = BC$. Then for every z, there are TC^0-Frege proofs of*

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

4.3 Some basic properties of iterative product

Lemma 25 *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.)

Lemma 26 *For every $z_1, ..., z_m$, and every $1 \le k \le m$, there are TC^0-Frege proofs of*

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

4.4 The Chinese-Reminder-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-Reminder-Theorem (a TC^0-Frege proof for the other direction is simpler).

Lemma 27 *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.)

We are now able to prove the following lemmas

Lemma 28 *For every z, there are TC^0-Frege proofs of*

$$PROD[z] = z.$$

Lemma 29 *For every z_1, z_2, there are TC^0-Frege proofs of*

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

4.5 A refutation for $DH_{P,g}$

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

Lemma 30 *For every $z_1, ..., z_m$, every $k \leq m-1$ and every fixed p, there are TC^0-Frege proofs of*

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

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

Lemma 31 *For every $z_{1,1}, ..., z_{m,m'}$ and every fixed p, there are TC^0-Frege proofs of*

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

Let us now show how to prove $g^{ab} modP = X^b modP$. We have to prove

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

But this follows by ,

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

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

5 Proofs of lemmas

Proof of Lemma 26 Recall that we have hard-coded the numbers u_j, such that $u_j mod q_j = 1$ and for all $i \neq j$, $u_j mod q_i = 0$. For all primes q_j dividing Q, and for all m, $1 \leq m \leq q_j$, we can verify the following statements: $[u_j \cdot m]_{q_j} = m$, and for all $i \neq j$, $[u_j \cdot m]_{q_i} = 0$. (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 2.4, but using r_{ij} only for i such that $k \leq i \leq m$.

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 $(g_j^{a_{i,j}}) mod q_j = r_{i,j}$, and also $a_{*,j}$ such that $(g_j^{a_{*,j}}) mod q_j = r_{*,j}$ 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 $g^{c'_j} mod q_j = r_j^{[1,...,k-1,[k,..,m]]}$ 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 26 is to show the following claim:

Claim 32 *For every j, $r_j^{[1,...,k-1,[k,...,m]]} = r_j^{[1,..,m]}$.*

The first step is to prove the following claim:

Claim 33 $PROD[z_k, .., z_m]_{q_j} = r_j^{[k,...,m]}$.

Claim 33 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 24; the third equality follows by Lemma 21, and Lemma 9. 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 32. 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 33, we have that $r_{*,j} = r_j^{[k,\dots,m]}$. Now $r_j^{[k,\dots,m]}$, 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 $z = r_{*,j} = r_j^{[k,\dots,m]}$), it follows that

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

□

Proof of Lemma 27 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 24 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. Now $Q = Q_t$. Assume by the induction hypothesis that

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

Denote, $D_R = div_{Q_{t-1}}[R]$, and $D_S = div_{Q_{t-1}}[S]$. Then by Lemma 18,

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

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 22

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

Since R, S are both lower than Q, it follows that D_R, D_S are both lower than q_t. Hence, by Claim 34 $D_R = D_S$. 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 34 *For every i, 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 $\{0, \dots, q_i\}$, and prove that $f(x)$ is onto the range $\{0, \dots, q_i\}$. Then, by applying the propositional pigeonhole principle, which is efficiently provable in TC^0-Frege, it follows that f is one to one.

□

Proof of Lemma 28 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 33, 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 27.

□

Proof of Lemma 29 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 27. By two applications of Lemma 23 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 2.4. By Claim 33, 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: $r_{1,i} = (g_i^{a_{1,i}}) mod q_i$, and $r_{2,i} = (g_i^{a_{2,i}}) mod q_i$, and therefore,

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

Also,

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

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

□

Proof of Lemma 30

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

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

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

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

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

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

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

The lemmas used for each equality in turn are: Lemmas 26,29, 23,29,26, and 26. □

Proof of Lemma 31 By an iterative application of the previous lemma.

□

6 Discussion

We have shown that TC^0-Frege does not have feasible interpolation, assuming that factoring 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 circuits that we use. It would be interesting to know to what extent our result holds in the uniform TC^0 proof setting.

7 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.

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

[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.

[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 to appear in *Journal of Symbolic Logic*.

[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.

[Im] Impagliazzo, R., Personal communication.

[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., "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.

[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.

[MP] Maciel, A., and Pitassi, T. "Normal forms for $AC^0[p]$-Frege proofs," Proceedings of STOC, 1997.

[Na] Naor, M., Personal communication.

[Pud] Pudlák , P., "Lower bounds for resolution and cutting planes proofs and monotone computations," *J. Symbolic Logic*, to appear.

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

[Pud3] Pudlák , P., Personal communication.

[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.