Electronic Colloquium on Computational Complexity, Comment 2 on Report No. 77 (2012)

# Succinct Interactive Proofs for Quantified Boolean Formulas

Edward A. Hirsch* Dieter van Melkebeek† Alexander Smal*

November 24, 2013

**Abstract**

In [4], it was claimed that the amount of communication in an interactive protocol for QBFormulaSAT can be bounded by a polynomial in the number of variables in the input formula. However, the proof was flawed. We give two correct proofs of this statement.

## 1 Setting

Chakraborthy and Santhanam [4] define SuccinctIP as the class of parameterized problems (an instance of a parameterized problem has the form $(x, 1^k)$, where $x \in \{0,1\}^*$ and $k \in \mathbb{N}$ is the parameter) that have two-sided error polynomial-time interactive protocols with the total number of bits of communication bounded by a polynomial in $k$.

QBFormulaSAT consists of true closed quantified Boolean formulas, where a formula can be any Boolean formula in the basis[1] $\{\vee, \wedge, \neg\}$. The parameter is the number of variables (that is, quantifiers).

Chakraborty and Santhanam claim the following statement in [4].

**Proposition 6.** QBFormulaSAT $\in$ SuccinctIP.

However, the proof in [4] is flawed. It uses Shen's [7] protocol for IP = PSPACE, which is a simplified version of Shamir's [6] original one. The problem arises in the penultimate steps of the protocol, which require sending high-degree polynomials for the quantifier-free part of the formula. These polynomials can contain too many coefficients as their degree is bounded by the size of the formula and not by a polynomial in the number of variables.

In the next two sections we suggest two different protocols that prove Proposition 6. The first protocol uses Rudich's protocol for PSPACE and provides a more general statement, namely that every language in PSPACE has a protocol in which the communication is bounded by a polynomial in its space complexity only. The second protocol uses a reduction by Santhanam and Williams [5] of QBFormulaSAT to QBCNFSAT, and then modifies Shen's protocol using an arithmetization somewhat similar to Shamir's original one. The main thing we have to keep an eye on is the length of the prover's messages, that is, the degree of the intermediate polynomials and the number of steps in the protocol.

*Steklov Institute of Mathematics at St.Petersburg, Russian Academy of Sciences, 27 Fontanka, 191023 St.Petersburg, Russia. Alexander Smal is partially supported by a grant of the president of RF MK-4108.2012.1.

†Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI 53706, USA. Partially supported by NSF grants CCF-1017597 and CCF-1319822.

[1] Actually, the basis does not matter either for the old or for the new proof. It could be any binary basis.



ISSN 1433-8092

It is worth noting that if a formula is replaced by a circuit, the resulting problem QBCircuitSAT demonstrates a kind of completeness property: if there is a SuccinctIP protocol for it, then there is a SuccinctIP protocol for every problem in alternating polynomial time (**AP**) with the number of alternating steps (bits) as a parameter. The protocol in Section 2 is in fact a public-coin interactive protocol such that not only the communication, but also the space used by the verifier is bounded by a polynomial in the space complexity of the original machine. Vice versa, every language accepted by a public-coin protocol has deterministic space complexity polynomial in the communication complexity and the space complexity of the verifier. It is unlikely that QBCircuitSAT can be accepted by such a protocol, because that would mean **P** is contained in polylogarithmic space, which is unknown and considered unlikely. The situation for general SuccinctIP protocols also remains open.

## 2 Protocol for low-space computations

In order to fix the issue with high-degree polynomials, we use a different interactive protocol than Shamir's or Shen's. We start with any PSPACE machine, consider its graph of configurations like in Savitch's theorem, and the prover convinces the verifier that there is a path reaching the accepting configuration. The polynomials in the protocol are multilinear. We follow the exposition of [1], in which the authors attribute the protocol to Steven Rudich, but we need a small tweak regarding the access to the input.

**Theorem 1.** *Let $M$ be an $s(x)$-space*[2] *deterministic Turing machine, and $L = \{x \mid M(x) = 1\}$, where $s$ is a polynomial-time computable*[3] *and polynomially bounded function. There is a polynomial-time interactive protocol for $L$ that uses at most $\mathrm{poly}(s(x) + \log|x|)$ bits of communication, public coins, and $\mathrm{poly}(s(x) + \log|x|)$ space for the (polynomial-time) verifier. Vice versa, any language posessing such a protocol can be accepted by a deterministic Turing machine using space $\mathrm{poly}(s(x) + \log|x|)$.*

*Proof.* Let $s'(x)$ denote the number of bits needed to describe a configuration of $M$ on input $x$, where a configuration contains the current state of the finite control, the contents of the work tapes accompanied by the work tape head positions, and the input tape head position (fine-tuning of the representation of this position is due later), but *not* the contents of the (read-only) input tape. We may assume that the head position is written in unary by considering the work tape as consisting of 2-bit blocks containing the stored bit and an indicator whether the head is present at that position. Note that $s'(x) = O(s(x) + \log|x|)$.

For two $s'(x)$-bit strings $\pi$ and $\pi'$, consider the predicate $\mathrm{PATH}_t(\pi, \pi')$ stipulating that there is a path of length $2^t$ from the configuration $\pi$ to the configuration $\pi'$. Wlog. we can assume that the accepting configuration is unique and that, once the accepting configuration is reached, the machine stays there forever. The prover starts with attempting to persuade the verifier that $\mathrm{PATH}_t(\alpha, \omega) = 1$, where $t = s'(x)$, $\alpha$ is the initial configuration (for the particular input $x$), and $\omega$ is the accepting configuration. In order to do this, they consider the arithmetization $P_t(\pi, \pi')$ of this predicate (a finite field-valued polynomial that has the same values as $\mathrm{PATH}_t$ on $\{0,1\}^{s'(x)} \times \{0,1\}^{s'(x)}$) that we will describe later. Then the protocol descends gradually from $P_{s'(x)}$ to $P_0$, the latter predicate being computable by the verifier itself.

[2]Note that $s$ may depend on the input and not just on its length.

[3]In fact, it suffices that the value of $s(x)$ is known to the verifier.

*The base arithmetization.* First of all let us describe the polynomial for $P_0$. This case needs additional attention. It *deviates* from [1], which only concerns the total polynomial space (both on input and work tapes) and hence cannot be directly applied to sublinear space (in particular, space bounded by a parameter). We need to separate the work tapes from the input tape. The polynomial for $\mathrm{PATH}_0$ is the sum of terms, one term per head position of the work tapes. Each term is conditioned on the event that the head is present (conditioning simply means multiplying by the corresponding bit $a_{2h}$). The term includes the multilinear "store polynomial"[4] expressing "$\bigwedge_{j\neq 2h+2,2h,2h-1,2h-2}(a_j = b_j)$" for the unchanged working memory, multiplied by a polynomial for the changed position (conditioned on the event that the head is present). In [1] this "change polynomial" uses $O(1)$ variables; however, once we separate the input tape from the working tape, the situation changes.

The "change polynomial" depends on the input. We can assume that the machine has oracle access to the input using a $\log|x|$-bit counter (stored in the bits $1\dots 2\log|x|$ of the configuration) so that its transition function can use the bit $x_i$, where $i$ is the current contents of the counter (note that the increment/decrement of the counter can be implemented using $\log|x|$ time and $O(1)$ space, so we can rewrite any machine this way). Then the answer to the oracle query is computed as the "input polynomial"

$$\sum x_i A_i, \tag{1}$$

where $A_i$ is the log-degree polynomial[5] stating that the string $a_1 a_3 \dots a_{2\log|x|-1}$ is the binary representation of $i$ (note that $x_i$ are constants for us now).

Once we can query the input, we can implement the transition function of $M$ by a multilinear "change polynomial" that has the following $O(1)$ variables: the bits of the current state, the head position represented by the bits $a_{2h}$, $a_{2h-2}$, $a_{2h+2}$, the currently affected memory location $a_{2h-1}$. The input bit (also needed by the transition function) is replaced by the polynomial (1).

The resulting polynomial $P_0$ is at most quadratic[6] in each variable.

*The other arithmetizations.* For $m \geq 0$, the polynomial $P_{m+1}$ is defined as

$$P_{m+1}(\pi,\pi') = \sum_{\sigma} P_m(\pi,\sigma) P_m(\sigma,\pi'), \tag{2}$$

where $\sigma$ ranges over all possible configurations of $M$. Note that this polynomial is an arithmetization of $\bigvee_\sigma \mathrm{PATH}_i(\pi,\sigma) \wedge \mathrm{PATH}_i(\sigma,\pi')$, and has degree at most four in each variable.

*The protocol.* Each round of the protocol reduces the task of checking the value of $P_{m+1}$ at a single point to checking the value of $P_m$ at a single point. A round consists of $s'(x)$ mini-rounds, each one eliminating the summation over one bit of the configuration $\sigma$ in (2), and an extra mini-round to reduce the verification of a product of two values of $P_m$ to a single one. We now detail each of the mini-rounds.

[4] Formally, $\prod_{j\neq 2h+2,2h,2h-1,2h-2}(a_j b_j + (1-a_j)(1-b_j))$. The write-up [1] has a typo here.

[5] Formally, $\prod_{j=1}^{\log|x|}((a_{2j-1} - i_j) + (1-a_{2j-1})(1-i_j))$.

[6] Why quadratic and not linear? This is because the counter is a part of the work tape, and the corresponding variables can appear in both the input polynomial and the other part of the "change polynomial".

*Summation mini-rounds.* For $\ell = 1, 2, \ldots$, the prover sends the coefficients of the polynomial $p_\ell$ (of degree at most four) in the outermost summation variable $\sigma_\ell$ such that

$$p_\ell(\sigma_\ell) = \left( \sum_{\sigma_{\ell+1}, \ldots} P_m(\pi, \sigma) P_m(\sigma, \pi') \right) \Bigg|_{\rho_{\ell-1}},$$

where $\rho_{\ell-1}$ is the substitution of the $\ell - 1$ previously instantiated variables built in the previous mini-rounds. Then the verifier checks that $p_\ell(0) + p_\ell(1) = v_{\ell-1}$, where $v_{\ell-1}$ is the value claimed in the previous mini-round (starting with the value $v_0 = P_{m+1}(\pi, \pi')$). After that, the verifier picks (and sends) a random value $r_\ell$ for $\sigma_\ell$, adds $\sigma_\ell = r_\ell$ to $\rho_{\ell-1}$ to obtain $\rho_\ell$, and the parties continue with the task of verifying that the value of the polynomial that now has one variable less, under the substitution $\rho_\ell$, equals $p_\ell(r_\ell)$ (which becomes $v_\ell$).

*Product mini-round.* When faced with the task of verifying $(P_m(\pi, \sigma) P_m(\sigma, \pi'))|_\rho = v$, where $\rho$ instantiates all bits of the configurations $\pi$, $\pi'$, and $\sigma$, the prover sends the coefficients of the linear function $p(z) = P_m(\pi + (\sigma - \pi)z, \sigma + (\pi' - \sigma)z)$, where $z$ is a new variable. The verifier then checks that $p(0)p(1) = v$, picks (and sends) a random value $r$, and the parties proceed to checking $(P_m(\pi + (\sigma - \pi)r, \sigma + (\pi' - \sigma)r))|\rho' = p(r)$, where $\rho' = \rho \cup \{z = r\}$.

*Putting everything together.* Provided the underlying field is sufficiently large, the protocol attains a small probability of error. Note that the degree of the intermediate polynomials cannot be more than four in each variable, because the base polynomial $P_0$ has degree at most two, and the degree raises only once, namely when the product $P_m(\pi, \sigma) P_m(\sigma, \pi')$ using a new configuration $\sigma$ is taken. Thus, the prover always sends at most five coefficients in each mini-round. The number of mini-rounds is $O(s'(x)^2)$.

*The inverse inclusion.* A protocol of the given type can be converted into a deterministic Turing machine using $\mathrm{poly}(s(x) + \log |x|)$ space. Indeed, the prover's messages and the random choices can be enumerated in $\mathrm{poly}(s(x) + \log |x|)$ space, and then the protocol's outcome can be easily computed. □

**Corollary 1.** *There is a SuccinctIP protocol for QBFormulaSAT.*

*Proof.* There is a Turing machine that solves QBFormulaSAT for formulas $F$ with $k$ quantifiers using $O(k \log |F|)$ space, because logarithmic space suffices for evaluating a formula on a particular assignment [3]. □

## 3 Protocol based on reduction to CNF

QBCNFSAT is the subset of QBFormulaSAT containing only formulas in conjunctive normal form. Santhanam and Williams [5] prove the following proposition.

**Proposition 1** ([5, Corollary 3.2])**.** *There is a polynomial-time reduction from QBFormulaSAT instances of size $s$ on $n$ variables to QBCNFSAT instances of size $O(s^4)$ on $n + O(\log s)$ variables.*

It remains to provide a SuccinctIP protocol for QBCNFSAT. Neither Shamir's [6] nor Shen's [7] protocol would do: in Shen's protocol the degree of the polynomial for the last quantifiers may be large because the arithmetization of $\wedge$ and $\vee$ both increase the degree, and Shamir's protocol is designed for "simple QBFs" instead of CNFs, and a straightforward reduction increases the number of variables by too much.

Instead, we use Shen's protocol with a different arithmetization that keeps the degree bounded in the case of CNFs. The arithmetization is close to Shamir's, but the roles of $\wedge$ and $\vee$ are interchanged. We will also need to work in a larger randomly chosen prime field, because our arithmetization can result in a *large* positive integer in the case of a false formula.

We include Shen's protocol for the sake of completeness so that one can easily check the number of transmitted bits.

**Theorem 2.** *There is a SuccinctIP protocol for QBCNFSAT.*

*Proof.* Let $\Psi$ be the input QBCNF formula:

$$\Psi = q_1 x_1 \; q_2 x_2 \; \ldots q_n x_n \; \psi(x_1, \ldots, x_n),$$

where $q_i \in \{\forall, \exists\}$ and $\psi$ is a CNF with $m$ clauses.

We rewrite $\psi$ as an arithmetic expression: we replace every clause $\bigvee y_i \vee \bigvee \neg z_j$ by $\prod(1-y_i)\cdot\prod z_j$ and take their sum, resulting in the polynomial $P_0$. Note that if $\psi(b_1, \ldots, b_n)$ holds true for a given $b_1 \ldots b_n \in \{0,1\}^n$, then $P_0(b_1, \ldots, b_n) = 0$; if $\psi(b_1, \ldots, b_n)$ is false, then $P_0(b_1, \ldots, b_n)$ equals the (positive) number of unsatisfied clauses. Note also that the polynomial $P_0$ is multilinear.

In order to arithmetize the quantifiers, we use the operators $Ax, Ex, Rx$ (where $x$ is a variable) that transform polynomials into different polynomials, namely

$$\begin{aligned} Ax \; p(x, \bar{y}) &= p(0, \bar{y}) + p(1, \bar{y}), \\ Ex \; p(x, \bar{y}) &= p(0, \bar{y}) \cdot p(1, \bar{y}), \\ Rx \; p(x, \bar{y}) &= (1 - x) \cdot p(0, \bar{y}) + x \cdot p(1, \bar{y}). \end{aligned}$$

Note that these operators keep the concept "zero = true", "nonzero = false" for 0/1-variables, namely, *as integers*, $Ax \; p(x, \bar{y}) = 0$ is zero iff $\forall x \in \{0,1\} \; p(x, \bar{y}) = 0$, $Ex \; p(x, \bar{y}) = 0$ is zero iff $\exists x \in \{0,1\} \; p(x, \bar{y}) = 0$, and $Rx$ does not change the value of the polynomial. We transform $\Psi$ into

$$\hat{\Psi} = Q_1 x_1 \; Rx_1 \; Q_2 x_2 \; Rx_1 \; Rx_2 \ldots Q_n x_n \; Rx_1 \ldots Rx_n \; P_0(x_1, \ldots, x_n),$$

where the quantifiers $q_i$ are replaced by the corresponding operators $Q_i$. (This $\hat{\Psi}$ is borrowed from Shen's protocol; for our arithmetization it would be enough to have $R$'s in front of the existential quantifiers only, because this is the only source of non-linearity.)

It is equivalent to say that $\Psi$ is true, and that $\hat{\Psi} = 0$.

There is still a small problem: the value of $\hat{\Psi}$ and the coefficients of the intermediate polynomials (which will later appear in the protocol) may be too large to be handled by a polynomial-time verifier. In order to keep them low, we will work in a finite field modulo a specially chosen prime. Namely, we will be happy with any reasonably large prime that does *not* divide $\hat{\Psi}$. Since $\hat{\Psi} \leq m^{2^n}$, $\hat{\Psi}$ has at most $2^n \log m \leq n2^n$ prime factors. By the Prime Number Theorem [2, Section 1.8], the number of primes up to $m$ is asymptotically equivalent to $m/\ln(m)$. It follows that the interval $[2^{n^2} \ldots 2^{2n^2}]$ contains $\Theta(2^{2n^2}/n)$ primes. Selecting a prime at random in this interval avoids the

prime factors of $\hat{\Psi}$ with overwhelming probability. In order to select a prime at random, one can repeatedly pick a random number in this interval and test it for primality; $O(2n^3)$ iterations result in a prime with probability at least $1 - e^{-n}$, and in a good prime, with probability close to that.

For the sake of completeness, we describe the protocol below.

*Handshake.* The verifier selects a prime number as described above and sends it to the prover. After that, all computations are performed modulo this prime (denote the corresponding field $\mathbb{F}$).

Then the parties proceed to verifying the claim $\hat{\Psi} = 0$.

*Intermediate steps.* At each step of the protocol the parties are faced with the equality $(Qx_i\hat{\Phi})|_\rho = v$, where $\hat{\Phi}$ is the part of $\hat{\Psi}$ appearing to the right of the operator $Qx_i$, $\rho$ is a substitution to all variables that appear in the operators to the left of the operator $Qx_i$, and $v \in \mathbb{F}$.

Let $\rho' = \rho$ if $Q \in \{A, E\}$, and $\rho'$ is obtained by dropping the value, say $w$, for $x_i$ from $\rho$ if $Q = R$. Note that $\hat{\Phi}|_{\rho'}$ is an at most quadratic univariate polynomial of the variable $x_i$. In order to persuade the verifier, the prover sends the three coefficients of this polynomial $p(x_i)$.

Then the verifier checks that the corresponding identity holds:

$$\begin{cases} v = p(0) + p(1), & \text{if } Q = \forall, \\ v = p(0) \cdot p(1), & \text{if } Q = \exists, \\ v = (1 - w) \cdot p(0) + w \cdot p(1), & \text{if } Q = R. \end{cases}$$

If the test fails, the verifier rejects.

Then the verifier picks a random value $r$ from the field $\mathbb{F}$ and sends it to the prover. The parties then proceed to verifying the identity $\hat{\Phi}|_{\rho''} = p(r)$ at the next step, where $\rho''$ is obtained by appending $x_i = r$ to the substitution $\rho'$.

*The final check.* Once there are no more operators left and all variables $x_i$ received their final values $r_i$, the verifier evaluates $P_0(r_1, \ldots, r_n)$ (in polynomial time) to compare with the final value it has. If the values are equal, it accepts, otherwise it rejects. □

**Corollary 2.** *There is a SuccinctIP protocol for QBFormulaSAT.*

*Proof.* Combine the reduction in Proposition 1 and the protocol in Theorem 2. □

## References


[1] D. M. Barrington and A. Maciel, *Basic Course on Computational Complexity.* Clay Mathematics Undergraduate Program, IAS/PCMI Summer Session 2000. Lecture 14. Web: `http://people.clarkson.edu/~alexis/PCMI/Notes/lectureB14.ps.gz`

[2] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers.* Oxford University Press, 2008.

[3] N. A. Lynch, *Log space recognition and translation of parenthesis languages.* Journal of the ACM 24: 583–590, 1977.

[4] C. Chakraborty, R. Santhanam, *Instance Compression for the Polynomial Hierarchy and Beyond*. ECCC TR12-077, 2012.

[5] R. Santhanam, R. Williams, *Uniform Circuits, Lower Bounds, and QBF Algorithms*, ECCC TR12-059, 2012.

[6] A. Shamir, $IP = PSPACE$, Journal of the ACM 39(4):869–877, 1992.

[7] A. Shen, *IP = PSPACE: simplified proof*, Journal of the ACM 39(4):878–880, 1992.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de