Electronic Colloquium on Computational Complexity, Report No. 17 (2014)

Short PCPs with projection queries

Eli Ben-Sasson*
Department of Computer Science
Technion — Israel Institute of Technology

Emanuele Viola†
College of Computer and Information Science
Northeastern University

February 9, 2014

Abstract

We construct a PCP for NTIME(2^n) with constant soundness, $2^n\text{poly}(n)$ proof length, and poly(n) queries where the verifier's computation is simple: the queries are a projection of the input randomness, and the computation on the prover's answers is a 3CNF. The previous upper bound for these two computations was polynomial-size circuits. Composing this verifier with a proof oracle increases the circuit-depth of the latter by 2. Our PCP is a simple variant of the PCP by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan (CCC 2005). We also give a more modular exposition of the latter, separating the combinatorial from the algebraic arguments.

If our PCP is taken as a black box, we obtain a more direct proof of the result by Williams, later with Santhanam (CCC 2013) that derandomizing circuits on n bits from a class C in time $2^n/n^{\omega(1)}$ yields that NEXP is not in a related circuit class C'. Our proof yields a tighter connection: C is an And-Or of circuits from C'. Along the way we show that the same lower bound follows if the satisfiability of the And of any 3 circuits from C' can be solved in time $2^n/n^{\omega(1)}$.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258. Email: eli@cs.technion.ac.il

†Supported by NSF grants CCF-0845003, CCF-1319206. Email: viola@ccs.neu.edu

ISSN 1433-8092

1 Introduction

It has long been known that solving satisfiability of circuits, or derandomizing probabilistic circuits implies new circuit lower bounds (for various exponential-time classes), see e.g. [KL80, IKW02]. In [Wil10] Williams gives an interesting instance of this phenomenon, where a non-trivial lower bound against a circuit class C follows from a satisfiability or derandomization algorithm for circuits of a related class C' that runs in time $2^n/n^{\omega(1)}$, where n is the number of variables of circuits in C'. It is an interesting question whether the approach based on satisfiability or the one based on derandomization should be pursed to obtain new circuit lower bounds.

The satisfiability approach – not the derandomization approach – has given non-trivial lower bounds [Wil11]. Moreover this approach has been tightened, by making C' closer to C, in [SW13, JMV13, Oli13], making it plausible that more lower bounds will be obtained. In fact, we will tighten it a bit more in this work. However, it is not clear how much this approach can be pushed. Do we believe that the satisfiability of (unrestricted) polynomial-size circuits can be solved faster than brute-force search? Even for seemingly simple problems such as MAX3SAT, no satisfiability algorithm better than brute-force search is known, despite attempts since a decade ago [Wil05]. Note that the MAX3SAT problem – given a 3CNF and an integer ℓ, is there an assignment that satisfies $\geq \ell$ clauses? – corresponds to the restricted class of depth-2 circuits known as MAJ $\circ$ AND_3: a Majority on And's on three variables. The lack of progress on MAX3SAT is an obstacle for obtaining new lower bounds from satisfiability.

A priori, the approach based on derandomization should apply more broadly, because most researchers indeed believe that derandomization is possible (and a long line of research has shown that indeed derandomization is possible based on lower bounds). Also, for several classes we have nontrivial derandomization algorithms but not satisfiability ones. For example, for the class mentioned above of MAJ $\circ$ AND_3 circuits a derandomization is given in [LVW93, Vio07]. Even when both types of algorithms are available, the speed of the derandomization one often outperforms that of the satisfiability one. For example, the running time for the derandomization of CNF, see [GMR13] for the latest, vastly outperforms that of satisfiability algorithms, cf. [Her11]. For another example, consider the class of poly-size, constant-depth circuits with Or, Not, and Parity gates (AC^0 with parity gates). To our knowledge, the best satisfiability algorithm is the one in [Wil11] which has running time 2^{n-n^ϵ}. By contrast, [FSUV13] derandomize these circuits in time $2^{n-n/\text{poly}\log n}$ (building on available lower bounds).

One advantage of satisfiability over derandomization is that the corresponding connection to lower bounds is simpler and incurs less overhead. To obtain lower bounds from derandomization one relies on probabilistically checkable proofs (PCP), specifically the somewhat intricate work by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [BGH$^+$05]. The intricacy of this work reflects on two aspects of the approach. First, to make it apply to restricted circuit classes such as ACC^0 or TC^0, previous to this work one needed a roundabout argument, provided by Santhanam and Williams [SW13], which actually relies on a subsequent PCP by Mie [Mie09] combining [BGH$^+$05] with Dinur's gap amplification [Din07]. Second, the indirect aspect of the argument is reflected in the overhead in the reduction. For example, to obtain a lower bound against circuits of depth d, one needed a derandomization algorithm for circuits of depth cd for a constant $c > 1$.

1.1 Our results

In this work we provide a variant of the PCP [BGH$^+$05] where the computation of the verifier is quite modest: Given randomness, the verifier computes its queries just by taking projections of the randomness, and the computation on the prover's answers is a 3CNF. The previous upper bound for these two computations was polynomial-size circuits.

Theorem 1.1 (Short PCPs with projection queries)**.** *Let M be an algorithm running in time $T = T(n) \geq n$ on inputs of the form (x, y) where $|x| = n$. Given $x \in \{0,1\}^n$ one can output in time* $\mathrm{poly}(n, \log T)$ *circuits* $\mathsf{Query} : \{0,1\}^r \to [2^r]^t$ *for* $t = \mathrm{poly}(r)$ *and* $\mathsf{Dec} : \{0,1\}^t \to \{0,1\}$ *such that:*

- **Proof length.** $2^r \leq T \cdot \mathrm{poly}\log T$,
- **Completeness.** *If there exists y such that $M(x, y)$ accepts then there exists a map $\pi : [2^r] \to \{0,1\}$ such that for any $z \in \{0,1\}^r$ we have $\mathsf{Dec}(\pi(q_1), \ldots, \pi(q_t)) = 1$ where $(q_1, \ldots, q_t) = \mathsf{Query}(z)$,*
- **Soundness.** *If no y causes $M(x, y)$ to accept, then for every map $\pi : [2^r] \to \{0,1\}$, at most $1/n^{10}$ fraction of the $z \in \{0,1\}^r$ have $\mathsf{Dec}(\pi(q_1), \ldots, \pi(q_t)) = 1$ where $(q_1, \ldots, q_t) = \mathsf{Query}(z)$,*
- **Complexity.** Query *is a projection (a.k.a. 1-local), i.e., each output bit of* Query *is one input bit, the negation of an input bit, or a constant;* Dec *is a 3CNF.*

The polynomial in the soundness item in Theorem 1.1 can be traded with the number t of queries.

There is a substantial literature that develops PCPs with optimized parameters. One focus of this literature has been to optimize the complexity of Dec. However typically these works do not produce PCPs of length quasilinear in T, and the complexity of Query is not optimized. Both these aspects are critical for our applications.

Remark 1.2 (Number of queries vs. Query complexity)**.** *Relaxing the complexity of* Query *to be a* $\mathrm{poly}(r)$*-computation allows to reduce the number of queries made to the oracle to a constant, while obtaining constant soundness [Mie09]. It is an interesting open problem to find the lowest complexity obtainable for* Query *in a PCP statement with proof length quasilinear in T, polylogarithmic verifier running time, and where soundness, alphabet, and number of queries are all constant. In particular, it is not clear to us if it is possible in such a case to have* Query *be a projection.*

Along the way we give a more accessible presentation of [BGH$^+$05]. Our presentation is modular and separates the combinatorial steps (given in Theorem 2.2) from the algebraic ones (given in §4).

Taking Theorem 1.1 as a black box, we eliminate the roundabout argument mentioned before from the result in [SW13] that derandomizing TC^0 circuits on n bits in time $2^n/n^{\omega(1)}$ implies that NEXP is not in TC^0. Also, Theorem 1.1 is a small variant on [BGH$^+$05], whereas as we mentioned [SW13] needs Mie's PCP [Mie09]. Finally, we also obtain the following alternative argument, which only uses the PCP result in [BGH$^+$05] as a black-box.

The alternative argument. Given as a black-box a PCP such as [BGH$^+$05], i.e., with the parameters as in Theorem 1.1 but where the complexity is replaced by polynomial-size circuits, we can construct a PCP where the verifier has low-complexity but makes adaptive queries to the proof. Specifically, we will rely on the prover to obtain the indices of our queries, and later query the prover at those indices and also verify the prover's computation, again with the help of the prover. This latter verification, as well as the computation Dec on the prover's answers, can be done by a 3CNF via a simple use of the Cook-Levin theorem – cf. Lemma 1.6.

Again, this alternative argument is sufficient to recover the TC^0 result in [SW13]. However, with Theorem 1.1 we obtain better parameters. Indeed, we seek very tight connections in the hope they will lead to progress on various challenges in computational lower bounds such as those mapped in [Vio13].

The concurrent work [Wil14] shows that the ability to count the number of satisfying assignments to circuits faster than brute-force search yields lower bounds against related circuits. This connection is used to obtain some new lower bounds. By our work the same lower bounds can be obtained from a satisfiability algorithm (using Theorem 1.5) or even a derandomization algorithm (using Theorem 1.4).

Next we state the tighter connections we obtain between derandomization and lower bounds. First we make a definition.

Definition 1.3. *Let C_n be a set of functions from $\{0,1\}^n$ to $\{0,1\}$. We say that C_n is* efficiently closed under projections *if functions in C_n have a $\mathrm{poly}(n)$-size description and given (the description of) a function $f \in C_n$, indexes $i, j \le n$, and a bit b, we can compute in time $\mathrm{poly}(n)$ the functions $\mathrm{not} f$, $f(x_1, \ldots, x_{i-1}, b \text{ XOR } x_j, x_{i+1}, \ldots, x_n)$, and $f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$, all of which are in C_n.*

Most of the standard classes have this property. For the theorem, the two occurrences of "poly(n)" above can be relaxed. We also use the notation $\wedge_{\mathrm{poly}(n)} \vee_3 C_{n+O(\log n)}$ to indicates the And of poly(n) Or of 3 functions from $C_{n+O(\log n)}$, all on the same n input bits.

Theorem 1.4 (Derandomization implies lower bounds, tightly)**.** *Let C_n be efficiently closed under projections.*

If the acceptance probability of functions of the form $h = \wedge_{\mathrm{poly}(n)} \vee_3 C_{n+O(\log n)}$ can be distinguished from being $= 1$ or $\le 1/n^{10}$ in Time($2^n/n^{\omega(1)}$), then there is a function f in E^{NP} such that $f_n \notin C_n$.

One can place f in NEXP if we replace $C_{n+O(\log n)}$ with $C_{\mathrm{poly}(n)}$ and reason as in [IKW01, Wil13, Wil11].

The first step of our more modular exposition of [BGH$^+$05] is a reduction to 3SAT that builds on [JMV13] (cf. [BCGT13a]) but achieves incomparable guarantees (Theorem 2.2). Using that, we can obtain the following connection between satisfiability algorithms and lower bounds.

Theorem 1.5 (Satisfiability implies lower bounds, tightly)**.** *Let C_n be efficiently closed under projections.*

If the satisfiability of functions $h = g_1 \wedge g_2 \wedge g_3$, where $g_i \in C_{n+O(\log n)}$ is in Time($2^n/n^{\omega(1)}$), then there is a function f in E^{NP} such that $f_n \notin C_n$.

The overhead to go from a satisfiability algorithm to a lower bound is evident from the theorem. The loss in size is a multiplicative factor $3 + o(1)$. Previous losses were polynomial [Wil10], or multiplicative by a larger constant [JMV13]. The loss in depth is 2 for circuits with fan-in 2. For unbounded fan-in (or even fan-in 3) circuits with And gates (or threshold) the depth loss is 1. Previous losses were 2 [JMV13, Oli13].

Recall that the best lower bound for an explicit function on n bits is $3n - o(n)$ (non-input) gates [Blu84] (cf. [DK11]). This seems to be the best known even for functions in E^{NP} (note the number of circuits of size $3n$ is superlinear, so one cannot easily diagonalize against them in E^{NP}). By Theorem 1.5, to obtain a function in E^{NP} of circuit complexity $3n$ one would need to solve satisfiability of a circuit with $3(3n)$ non-input gates and n inputs – ignoring lower-order terms. The Cook-Levin theorem reduces this to a 3SAT instance on $9n + n = 10n$ variables. So one would need to solve 3SAT in deterministic time c^n for any $c < 2^{1/10} = 1.07\ldots$ The current record is $c = 1.33\ldots$ [MTY11], cf. [Her11].

1.2 Techniques

Ideas behind the proof of Theorem 1.1. We start with the PCP in [BGH$^+$05] and follow its proof closely. There are two computations of the verifier in this PCP that we need to optimize. The first — Query — is taking the input randomness to the queries, which we call *preprocess*. The second — Dec — is the computation on the prover's answers, which we call *postprocess*. We discuss them separately.

Postprocess: It is a common experience in theoretical computer science research, to study at length an intricate proof in the reckless hope of optimizing parameters, only to be surprised by the late realization that a trivial, sweeping argument takes the complex proof as a black-box and gets a pretty good parameter optimization, too.

Lemma 1.6 (Making Dec a 3CNF)**.** *Suppose Theorem 1.1 holds except that* Dec *is an unrestricted circuit of size* $\mathrm{poly}(r)$*. Then Theorem 1.1 holds as stated.*

Proof. By the Cook-Levin theorem we reduce Dec to a 3CNF with $\mathrm{poly}(r)$ variables and terms. The verifier will ask the prover for an additional $\mathrm{poly}(r)$ queries to obtain the satisfying assignments for this 3CNF corresponding to the input randomness. On input z, these queries are of the form (z, i), where i is an $O(\log r)$-bit index to a variable in the 3CNF. The proof contains the values of the variables in the 3CNF that verify the computation on the outputs of the queries that are made by the verifier on input z. □

This general technique shows that we can always make the postprocess a 3CNF as long as we allow for poly(r) queries. Using it, there is no benefit in reducing the number of queries to a constant.

Preprocess: This is in turn comprised of two parts, acting in parallel, known as "algebraic constraint satisfaction" and "low-degree testing", and in this work we offer a clear separation between the two. In the first part we reduce the succinct constraint satisfaction problem (CSP) associated with verifying the M accepts x in T steps, to an algebraic CSP (ACSP) problem, one stated as a question about equality of polynomials. We offer a definition of ACSP that is algebraically cleaner than [BS08] and following works (e.g., [BGH$^+$05, BCGT13b]). In particular, previous definitions included degree bounds on the "assignment polynomial" and involved a "zero-testing" problem. In contrast, Definition 4.1 defines a satisfying assignment as one that causes a polynomial to vanish, and degree-bounds are dealt with by the separate low-degree testing part, discussed later. We now elaborate on how we obtain efficient preprocessing in each of the two parts.

In the ACSP part, our verifier simply selects a random field element α, generates poly(r) queries to the prover where the ith query is $\alpha + \sigma_i$ where σ_i is fixed and independent of α, cf. §4. Each such query can be verified to be a projection. To reach this simple form of preprocessing we use a modular reduction from the combinatorial succinct CSP captured by Theorem 2.1 to the succinct ACSP stated in Theorem 4.2. The mid-point between the combinatorial and algebraic settings is given in Theorem 2.2. In it we reach a 3CNF formula with $\approx T$ clauses where each clause (i.e., the three variables of the clause and their polarities) can be computed by a simple XOR operation. Since XOR is addition in fields of characteristic 2, irrespective of the basis chosen for them, we get a simpler ACSP than [BS08, BGH$^+$05] albeit one that has a super-constant number of variables.

Turning to the second part, low-degree testing, we use auxiliary information in form of a PCP of Proximity (PCPP) [BSGH$^+$06, DR06]. This part is essentially from [BS08] and regrettably remains an intricate step of the proof. The answers to queries of the verifier in [BS08] can be seen as arranged in the nodes of a tree. The query at each node is indeed a projection. However, the verifier uses part of its input randomness to select a path in this tree, reaching a leaf, then possibly redirects the query to a node higher up in the tree. This computation is more complicated than just a projection. Here we use the following simple, key idea. The path from root to leaf is determined by only $O(\log r)$ of the verifier's r input bits (cf. Claim 6.3). Additionally, the process of redirecting a query from a leaf to a node elsewhere in the tree is also determined by only $O(\log r)$ input bits (cf. Claim 6.4). Instead of following the path, we let the verifier query every possible endpoint. This multiplies the number of queries by a factor poly(r), which we can afford. We delegate the task of picking the right query to the postprocess.

One more complication is that each of the two parts needs to be combined with a randomness-efficient hitter to achieve constant soundness. Using e.g. Cayley expanders built from small-bias sets, this step is again just a projection (cf. §4.3).

Ideas behind the proof of Theorem 1.5. A natural idea is to improve the previous constant-locality result [JMV13] to locality 1. But this may not be possible. Instead, we show how to reduce arbitrary computation to a polynomial number of 3CNF formulae, each of which has locality 1. By enumerating over these 3CNF, and running the satisfiability algorithm on each of them, we get the result. This idea is similar to the one described above to make [BS08] a projection: after reading a logarithmic number of bits, the rest of the computation becomes just a projection.

Open problems. Improve Theorem 1.4 to have the same overhead as Theorem 1.5.

Organization. In §2 we give a variant of the reduction of non-deterministic time to 3SAT given in [JMV13]. Using that and Theorem 1.1 as a black-box, in §3 we give the proofs of theorems 1.4 and 1.5. In §4 we give the new exposition of [BGH$^+$05] based on the reduction in §2. By Lemma 1.6, we only need to verify that Query can be implemented by projections. §4 states two main claims, and then proves Theorem 1.1 assuming those. The main claims are in turn discussed in §5, 6.

2 A combinatorial reduction to 3SAT

Our starting point is the following result from [JMV13].

Theorem 2.1 ([JMV13]). *Let M be an algorithm running in time $T = T(n) \geq n$ on inputs of the form (x, y) where $|x| = n$. Given $x \in \{0,1\}^n$ one can output a circuit $D : \{0,1\}^\ell \to \{0,1\}^{3v+3}$ in time $\mathrm{poly}(n, \log T)$ mapping an index to a clause of a 3CNF ϕ in v-bit variables, for $v = \Theta(\ell)$, such that*

1. *ϕ is satisfiable iff there is $y \in \{0,1\}^T$ such that $M(x, y)$ accepts, and*
2. *For any $r \leq n$ we can have $\ell = \max(\log T, n/r) + O(\log n) + O(\log\log T)$ and each output bit of D is a decision tree of depth $O(\log r)$.*

Note that for $T = 2^n$ and $r = O(1)$ this gives a 3CNF with $T \mathrm{poly}\log T$ clauses such that each clause can be computed from its index by a function with constant locality.

We need an incomparable variant of the latter. We enlarge the locality to $O(\log n)$, but at the same time there are only $O(\log n)$ input bits that affect more than 1 bit. If we fix these bits, the rest of the computation is just a bit-wise xor.

Theorem 2.2. *Let M be an algorithm running in time $T = T(n) \geq n$ on inputs of the form (x, y) where $|x| = n$. Let $\ell_1 = \log T$. For some $\ell_2 = O(\log\log T) + O(\log n)$ the* *following is true. Given $x \in \{0,1\}^n$ one can output in time* $\mathrm{poly}(n, \log T)$ *six circuits (of size* $\mathrm{poly}(n, \log T)$*)*

$$S_i : \{0,1\}^{\ell_2} \to \{0,1\}^{\ell_1+\ell_2},$$
$$b_i : \{0,1\}^{\ell_2} \to \{0,1\},$$

for $i = 1, 2, 3$, such that:

Let ϕ_x be the 3CNF with $2^{\ell_1+\ell_2} = T\mathrm{poly}(n, \log T)$ clauses (and variables) whose $(\alpha, \beta) \in \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2}$ clause contains variables

$$V_i = (\alpha, \beta) \oplus S_i(\beta)$$

and corresponding sign bits

$$b_i = b_i(\beta),$$

where $i = 1, 2, 3$ and $\oplus$ is bit-wise xor. Then ϕ_x is satisfiable iff there is $y \in \{0,1\}^T$ such that $M(x, y)$ accepts.

Note that in the case $T = 2^{O(n)}$ each output bit of D depends only on $|\beta| + 1 = O(\log n)$ bits of the input.

The next proof heavily builds on previous works. We give a sketch that highlights the tiny changes from previous proofs, and to work out parameters. The closest previous proof is [JMV13], to which we also refer for a discussion of other related works.

Proof sketch. Without loss of generality the algorithm is implemented by a random-access Turing machine running in time $T' = T\mathrm{poly}\log n$ and only using memory cells at indexes $\leq \mathrm{poly}(T)$ (see e.g. [NEU12] for details).

Consider a circuit-sat instance where the circuit first guesses a computation trace consisting of T'

configurations of size $O(\log T)$ each; and then the circuit checks its validity and acceptance. The validity check consists of two separate checks. The first is the check of the consistency of the transition function of the machine, assuming that memory reads are correct. The second is the check of the consistency of the memory reads and writes. The trace is valid if and only if both checks pass. These two checks are implemented in a similar fashion; we only describe the second.

Consider a matrix of $r \times T'$ configurations, where $r = \mathrm{poly}\log T$. We use α to index a column in this matrix. (This actually gives $|\alpha| = \ell_1 = \log T' = \log T + O(\log\log T)$, but the low-order summand can be swallowed in ℓ_2.) We use β to index a row, and the gates within the subcircuits discussed next.

The first row is the computation trace mentioned above that the circuit guessed. For every $t = 1, \ldots, T$ we have a $\mathrm{poly}(n, \log T)$-size subcircuit which checks the pair of configurations (C, C') at positions $(1, t)$ and (r, t) in the matrix, i.e., in the same column but at antipodal rows. This subcircuit verifies that either C' accesses the same memory cell of C and has the timestamp of C plus one, or C' accesses a memory cell with index greater than that of the cell accessed by C, or – the wrap-around condition – it does nothing if C' is the configuration with timestamp 0. If all these checks pass then for every t the configuration at position (t, r) is the one that comes next the configuration at position $(t, 1)$ in the order given by memory location accessed breaking ties by timestamp. The subcircuit then verifies consistency of the memory read and write in C and C'. In particular it verifies that cells read for the first time are blank, and that the cells $1, \ldots, n$ read for the first time contain the input x. Note that the latter is possible because our circuits have size $\geq n$ and are built with knowledge of x.

Observe that these subcircuits operate independently on each column, and are identical across columns. Their connections depend only on the row index and an index to one of their gates. By including these two indexes inside β, these connections can be computed in the required format.

It remains to discuss connections across columns, which are needed to move configurations around to put them in the right order. For this we use routing networks such as Beneš', which are a simple composition of butterfly networks. The index of a neighbor in column α is obtained by xoring α with a string (of Hamming weight 1) which only depends on the row, which in turn is part of β. This leads to the desired format for V_i. The implementation of the routing network also needs a simple gadget to swap two configurations depending on a nondeterministic bit. This gadget is the same for every column and row. From this it follows that the V_i and b_i are in the desired format. □

3 Lower bounds from fast algorithms

In this section we prove theorems 1.4 and 1.5. First we restate the theorem.

Theorem 1.5 (Satisfiability implies lower bounds, tightly)**.** *Let C_n be efficiently closed under projections.*

If the satisfiability of functions $h = g_1 \wedge g_2 \wedge g_3$, where $g_i \in C_{n+O(\log n)}$ is in Time($2^n/n^{\omega(1)}$), then there is a function f in E^{NP} such that $f_n \notin C_n$.

The proof follows [JMV13]. The most important change is that we use Theorem 2.2, and we *enumerate* over the β in its statement. The key observation is that for any fixed β, the reduction is only computing xor which can be hardwired with no loss in resources. This enumeration is feasible because $|\beta| = O(\log n)$. To get better constants we also work with unary languages.

Proof of Theorem 1.5. Suppose that every function in E^{NP} belongs to C_n when restricted to inputs of length n. Let L be a unary language in $\mathrm{NTime}(2^n) \setminus \mathrm{NTime}(o(2^n))$ [Coo73, SFM78, Zák83]. Consider the E^{NP} algorithm that on input $x' \in \{0,1\}^{O(\log n)}$ and $i \leq 2^n \mathrm{poly}(n)$ computes $x = 1^{x'}$, the 3CNF ϕ_x corresponding to L through Theorem 2.2, computes its first satisfying assignment if one exists, and outputs its ith bit. By assumption, on inputs of length $m = n + O(\log n)$ this function is in C_m. Also, by assumption, if we hardwire x' the resulting function still belongs to C_m. Call this function g_x.

We contradict the assumption on L by showing how to decide it in $\mathrm{Ntime}(o(2^n))$. Let D, S_i, α, β, V_i and b_i be as in Theorem 2.2. Consider the algorithm that on input $x = 1^n$

guesses g_x. Then it constructs the function g'_x that operates as follows. The input is that of D. Then it connects three copies of g_x to the output variables V_i. Further, the output of the ith copy is negated and then xored with b_i. And finally an And is taken. Call g'_x this new function (which may not belong to any C_n). Note that $g'_x(i) = 1$ iff the ith clause is not satisfied (by satisfying assignment g_x). So by determining the satisfiability of g'_x we can determine if $x \in L$ or not.

The satisfiability algorithm enumerates over all poly(n) choices for β. For each fixed β, the b_i are determined, and the remaining computation to obtain the V_i is an xor by $S_i(\beta)$. All this can be hardwired into g'_x in time poly(m), because C_m is efficiently closed under projections. For every i this gives a new function $g_i \in C_m$. There remains to solve the satisfiability of $g_1 \wedge g_2 \wedge g_3$. The latter can be done in time $2^m/m^{\omega(1)}$ by assumption. So overall the running time is $\text{poly}(n, m)2^m/m^{\omega(1)} = 2^n/n^{\omega(1)} = o(2^n)$. □

Theorem 1.4 (Derandomization implies lower bounds, tightly)**.** *Let C_n be efficiently closed under projections.*

If the acceptance probability of functions of the form $h = \wedge_{\text{poly}(n)} \vee_3 C_{n+O(\log n)}$ can be distinguished from being $= 1$ or $\leq 1/n^{10}$ in Time($2^n/n^{\omega(1)}$), then there is a function f in E^{NP} such that $f_n \notin C_n$.

Proof sketch. Proceed as the proof of Theorem 1.5, but let ϕ_x be instead of the 3CNF produced by Theorem 2.2 the constraint satisfaction problem corresponding to our main theorem, 1.1. As before, we obtain a function g'_x that on input i determines if the ith constraint is satisfied. (To show that the complexity of this function is as desired, we merge the Not gates of the 3CNF corresponding to Dec with the circuits in $C_{n+O(\log n)}$, using the closure of the class.) Thus, approximately determining how many constraints are satisfied amounts to approximately determinining the number of satisfying assignments to g'_x. □

4 Proof of Main Theorem 1.1

In this section we prove Theorem 1.1. To do so we recall that the succinct verifier of [BS08, BGH+05] contains two sub-verifiers, the first one checks an Algebraic Constraint Satisfaction Problem (ACSP) problem and the second verifies proximity to Reed-Solomon codes. To prove the main theorem we need to show that both can be computed with low computational complexity. In §4.1 we discuss the efficient verification of ACSP. In §4.2 we state the efficiency of the proximity tester. In §4.3 we state the needed derandomization tools for boosting soundness via repetition and in §4.4 we complete the proof.

4.1 Algebraic CSP

The following definition is a variant on that of a univariate ACSP appearing in [BS08, BGH+05] (cf. [Mie09, BCGT13b]). It slightly differs as it requires an assignment which is a *pair* of univariate polynomials (previous works required a single univariate polynomial) but its definition of satisfiability (1) is cleaner from an algebraic point of view.

Definition 4.1 (Algebraic CSP). *A* univariate ACSP *(or, simply, an ACSP) instance ψ is a quadruple $\psi = (\mathbb{F}, J, \mathcal{V}, Q)$ where*

- *$\mathbb{F}$ is a finite field*
- *J is a finite set of indices*
- *$\mathcal{V} = \{V_j \mid j \in J\}$ is a set of univariate polynomials ($V_j \in \mathbb{F}[X]$) indexed by J*
- *$Q = Q(X, Z, \mathcal{Y} = \{Y_j \mid j \in J\})$ is a multi-variate polynomial over variables X, Z and the set of variables $\mathcal{Y}$ which, like $\mathcal{V}$, is indexed by J.*

An assignment *to ψ is given by two polynomials $A, B \in \mathbb{F}[X]$. We say (A, B)* satisfies *ψ if*

$$Q\left(X, A(X), B(X), \{Y_j \leftarrow A(V_j(X)) \mid j \in J\}\right) \equiv 0 \tag{1}$$

where equality is over the ring $\mathbb{F}[X]$ and $Y_j \leftarrow A(V_j(X))$ means substituting Y_j by $A(V_j(X))$.

The first part in the proof of Theorem 1.1 is

Theorem 4.2. *Let M be an algorithm running in time $T = T(n) \geq n$ on inputs of the form (x, y) where $|x| = n$. There exists an integer c and another algorithm that given $x \in \{0,1\}^n$ and integer $a \geq 2$, outputs in time* $\text{poly}(n, \log T, a)$ *an ACSP instance $\psi = (\mathbb{F}, J, \mathcal{V}, Q)$ as in Definition 4.1. This ACSP is satisfiable iff there is $y \in \{0,1\}^T$ s.t. $M(x, y)$ accepts, and has the following features:*

- **Characteristic** 2. *$\mathbb{F}$ is the field of size $2^{\ell+a}$ where $\ell = \log(n+T) \cdot O(\log\log(n+T))$.*
- **Preprocessing.** *$|J| = \text{poly}(\log T, n)$ and $\deg(V_j) \leq 1$ for all $j \in J$. Moreover, V_j is of the form $V_j(X) = X + \gamma_j$ where $\gamma_j \in \mathbb{F}$.*
- **Degree bounds.** *If Q is satisfiable then it is satisfied by polynomials A and B of degrees at most 2^ℓ and $2^{\ell+c}$ respectively.*
- **Postprocessing.** *Q is computed by an arithmetic circuit over $\mathbb{F}$ of size at most* $\text{poly}(\log T, n)$.

Note that by the Preprocessing feature, each V_j is computed by a projection as per the last bullet of Theorem 1.1.

Remark 4.3 (Constant size J). *The construction of [BGH⁺05, BCGT13b] has constant size J (i.e., constant query complexity). However, this comes at the price of a more complicated construction, resulting in preprocessing that is not necessarily a projection. Also note that even if J were to be constant, proximity-testing (discussed in next section) requires super-constant query complexity.*

Following [BGH$^+$05, BCGT13b], the proof idea is to "arithmetize" the circuit D from Theorem 2.1 into a polynomial Q: Inputs and outputs of this arithmetized version are elements of a field $\mathbb{F}$ of size roughly 2^ℓ and an assignment is supposed to be the interpolation (or "low-degree extension") of the assignment to the variables. We denote the assignment by $A(X)$. (The role of the polynomial B is to solve the "zero-testing" problem and will be explained later.) Since the definition above allows A to be any polynomial over $\mathbb{F}$ we need to add a set of "range-checking" constraints, not appearing in D, that force A to evaluate only to $\{0, 1\}$ on the domain of interest.

4.2 Reed Solomon PCP of Proximity

We view a linear error correcting code C as a set of functions $C = \{w : L \to \mathbb{F}\}$ from a finite domain L of size n (the code's blocklength) to a finite field $\mathbb{F}$.

Definition 4.4 (Succinct PCPP for a linear code)**.** *Let $C = \{w : L \to \mathbb{F}\}$ be a linear error correcting code over finite field $\mathbb{F}$. A PCP of proximity (PCPP) for C is a set of functions, one for each codeword of C, denoted $\Pi = \{\pi_w : L' \to \mathbb{F}_q \mid w \in C\}$, where $L \cap L' = \emptyset$. A* succinct PCPP-verifier *for (C, Π) is a pair $(\mathsf{Query}, \mathsf{Dec})$ where*

- **Preprocessing** $\mathsf{Query} : \{0, 1\}^r \to \mathbb{F}^t \times (L \cup L')^q$. *In what follows we denote by $x_1^{(R)}, \dots, x_{t+q}^{(R)}$ are the outputs of $\mathsf{Query}(R)$ on input $R \in \{0, 1\}^r$*
- **Postprocessing** Dec *is an arithmetic circuit over $\mathbb{F}$ with $t + q$ inputs*

The proof length *of this PCPP is $|L'|$, its* alphabet *is $\mathbb{F}$, the* randomness complexity *is r,* query complexity *is q and* decision complexity *is $|\mathsf{Dec}|$. We say C has a PCPP system with* soundness *s for* proximity parameter *δ_0 if there exists a PCPP Π for C satisfying:*

- *Completeness: If $w \in C$ then*

$$\Pr_{R \in \{0,1\}^r}[\mathsf{Dec}(x_1^{(R)}, \dots, x_t^{(R)}, y_1, \dots, y_q) = 0] = 1$$

  where $y_i = (w \circ \pi_w)(x_{i+t}^{(R)})$ is the answer given to query x_{i+t}

- *Soundness: If w is δ_0-far from C then for any $\pi : L' \to \mathbb{F}$,*

$$\Pr_{R \in \{0,1\}^r}[\mathsf{Dec}(x_1^{(R)}, \dots, x_t^{(R)}, y_1, \dots, y_q) \neq 0] \geq s$$

We find it helpful to define Reed-Solomon codes in a somewhat non-standard way, using fractional degree δ instead of regular degree.

Definition 4.5 (RS-codes)**.** *For $\mathbb{F}$ a finite field, $L \subset \mathbb{F}$ and constant $\delta \in (0, 1]$*

$$\mathrm{RS}[\mathbb{F}, L, \delta] = \{p : L \to \mathbb{F} \mid \deg(p) < \delta |L|\}$$

where $\deg(p)$ *is the minimal degree of a polynomial $P(X)$ whose evaluation over L is p.*

The second part in the proof of Theorem 1.1 is given next.

Theorem 4.6 (Preprocessing efficient verifier for RS codes)**.** *Let $\mathbb{F}$ be a finite field of characteristic* 2 *and $L \subseteq \mathbb{F}$ be an $\mathbb{F}_2$-affine space of dimension k and let $a \geq 3$ be an integer. For any proximity parameter $\delta_0 > 0$ there exists a succinct PCPP system for* $\mathrm{RS}[\mathbb{F}, L, 2^{-a}]$ *with soundness $1/k^c$ for a positive constant c independent of k, satisfying*

- **Proof length** *is* $2^k \cdot \mathrm{poly}(k)$ *and alphabet is* $\mathbb{F}$
- **Query complexity** *is $k^{c'}$ for positive constant c' independent of k*
- **Preprocessing:** Query *is a projection according to the last bullet of Theorem 1.1*
- **Postprocessing:** $|\mathsf{Dec}| = \mathrm{poly}(k)$

We point out that all points but for preprocessing were proved in [BS08, BGH$^+$05]. Therefore our proof of the theorem in Section 6 will focus only on preprocessing.

4.3 Randomness efficient sampling via ϵ-biased sets

For boosting soundness in an efficient way, i.e., one that preserves 1-local preprocessing, we need to use samplers based on ϵ-biased sets. After recalling the necessary information about them we complete the proof of Theorem 1.1.

The following lemma is a well-known corollary of the expander mixing lemma (cf. [ASE92]) and the observation that a Cayley graph generated by an ϵ-biased set has normalized second eigenvalue at most ϵ (cf. [NN90]). We state it without proof.

Lemma 4.7 (ϵ-biased hitting sets)**.** *Let G be a Cayley graph over vertex set $\{0,1\}^n$ generated by an ϵ-biased set S. Then for any $V \subset \{0,1\}^n$ of density $\mu = |V|/2^n$, the set U of vertices that have no edge to V has density*

$$\frac{|U|}{2^n} \leq \frac{\epsilon^2}{\mu}$$

We also use the existence of efficiently constructible ϵ-biased sets (cf. [NN90, AGHP92]).

Lemma 4.8 (ϵ-biased sets)**.** *Given $\epsilon \in (0,1)$ and integer n there exists a deterministic algorithm running in time* $\mathrm{poly}(n, 1/\epsilon)$ *that outputs an ϵ-biased set S. Consequently, $|S| \leq$* $\mathrm{poly}(n, 1/\epsilon)$*.*

4.4 Proof of Theorem 1.1

Recall the proof of [BS08, BGH$^+$05]. On input x one invokes Theorem 4.2 with integer $a = c + 10$ to obtain an ACSP instance as stated there (we use the notation from that theorem). Verifier expects a PCP proof containing *(i)* a function $p_A : \mathbb{F} \to \mathbb{F}$ with PCPP π_A for $\mathrm{RS}[\mathbb{F}, \mathbb{F}, \frac{2^\ell}{|\mathbb{F}|}]$ where $\mathbb{F}$ is the finite field of size 2^ℓ and $\ell = \log(n + T) \cdot O(\log\log(n + T))$; The function p_A is supposed to be the evaluation of A on $\mathbb{F}$, and *(ii)* a function $p_B : \mathbb{F} \to \mathbb{F}$

with PCPP π_B for RS$[\mathbb{F}, \mathbb{F}, \frac{2^{\ell+c}}{|\mathbb{F}|}]$. Now verifier applies *(i)* the RS-PCPP sub-verifier on each of (p_A, π_A) and (p_B, π_B), and *(ii)* checks that (1) in Definition 4.1 holds for a random choice of $\xi \in \mathbb{F}$, and using p_A, p_B as proxies for A and B; all of these tests reuse randomness. If all tests pass, verifier accepts, else she rejects. It is shown in [BS08] that this PCP has perfect completeness, soundness $1/\text{poly}(n)$, query and decision complexities $\text{poly}(n, \log T)$, proof length $(n+T) \cdot \text{poly}\log(n+T)$ and randomness complexity $\log \ell + O(\log\log \ell)$. Thus, what is left is to argue is that we can repeat the process $\text{poly}(n)$ times to boost soundness to $1 - 1/\text{poly}(n)$ and do this while maintaining the preprocessing as a 1-local computation, or projection, and the decision complexity of $\text{poly}(n, \log T)$ which can then be converted to a 3CNF of size $\text{poly}(n, \log T)$ using the Cook-Levin Theorem (cf. Claim 1.6).

The preprocessing of a single invocation of the verifier described above is a 1-local computation, as argued for the ACSP verifier in Theorem 4.2 and for the RS-PCPP verifier in Theorem 4.6. Let $S \subset \{0,1\}^r$ be an ϵ-biased set for a suitably small $\epsilon = 1/\text{poly}(n)$ to be fixed later. Lemma 4.8 implies $|S| = \text{poly}(n)$. Given randomness $R \in \{0,1\}^r$ we invoke the verifier above (which has soundness $1/\text{poly}(n)$) on $R \oplus s$ for each $s \in S$, where $\oplus$ denotes xor. The transformation $R \mapsto R \oplus s$ is 1-local because s is fixed so the new verifier still has a 1-local preprocessing computation. Postprocessing and query complexity can be seen by inspection to be $\text{poly}(n)$ and soundness is now boosted to $1 - 1/\text{poly}(n)$ due to Lemma 4.7. Indeed, let $V \subset \{0,1\}^r$ be the set of randomness strings that cause the single-invocation verifier to reject, and let $U \subset \{0,1\}^r$ be the "bad" set of R such that $R \oplus s \notin V$ for every $s \in S$. Lemma 4.7 implies that the density of U in $\{0,1\}^r$ can be reduced to at most $1/n^{c_1}$ by fixing $\epsilon = 1/n^{c_2}$ for a constant c_2 depending only on c_1.

This completes the proof of Theorem 1.1.

5 Proof of Theorem 4.2

5.1 Algebraic preliminaries

In our proof we need to use selector polynomials and their efficient construction, defined next.

Definition 5.1 (Vanishing and selector polynomials)**.** *For $S \subset \mathbb{F}$ the* vanishing polynomial *of S is $\mathsf{Zero}_S(X) = \prod_{s \in S}(X - s)$. In words, it is the unique monic polynomial of degree $|S|$ whose set of roots, counted with multiplicitiy, is precisely S. For $S' \subset S$ we define the* selector polynomial *of S' in S to be*

$$\mathsf{Sel}_{S' \subset S}(X) \triangleq \mathsf{Zero}_S(X)/\mathsf{Zero}_{S'}(X)$$

noticing it has degree $|S| - |S'|$, vanishes on $S \setminus S'$ and is nonzero on S'. We use $\mathsf{Sel}_{\beta \in S}(X)$ as a shorthand for $\mathsf{Sel}_{\{\beta\} \subset S}(X)$.

Lemma 5.2 (Succinct affine selector polynomials)**.** *If $S' \subset S$ are two $\mathbb{F}_q$-affine subspaces inside $\mathbb{F}_{q^N}$ of respective dimensions $d' < d$, then $\mathsf{Sel}_{S' \subset S}(X) = \mathsf{Zero}_S(X)/\mathsf{Zero}_{S'}(X)$ is computed by an arithmetic circuit over $\mathbb{F}$ of size* $\text{poly}(d)$*.*

Proof. Follows from the fact that $\mathsf{Zero}_S(X), \mathsf{Zero}_{S'}(X)$ are linearized polynomials, hence each have d, d' nonzero coefficients respectively. □

5.2 Proof of Theorem 4.2

Notation For $\mathbb{F}$ a field of characteristic 2 and $B = (\beta_1, \dots, \beta_n)$ linearly independent over $\mathbb{F}_2$ and $\alpha \in \mathbb{F}$, let $\text{span}(B)$ be the span of B. For $S, T \subset \mathbb{F}$ let $S+T = \{\alpha + \beta \mid \alpha \in S, \beta \in T\}$. For $\alpha \in \mathbb{F}$ let $\alpha + S = \{\alpha\} + S = \{\alpha + x \mid x \in S\}$, noticing $\alpha + \text{span}(B)$ is the affine shift of $\text{span}(B)$ by α.

Proof of Theorem 4.2. The proof goes by arithmetizing the circuits and constraint satisfaction problem of Theorems 2.1, 2.2, so we use the parameters and notation stated there.

Associate $\{0, 1\}$ with elements of $\mathbb{F}_2$ in the natural way. Let $N > \ell$, let $\mathbb{F}$ be the field of size 2^N and let $\zeta_1, \dots, \zeta_N$ be a basis for it.

Embedding the inputs to D in $\mathbb{F}$ Embed the first ℓ_1 inputs of D into $H_1 = \text{span}(\zeta_1, \dots, \zeta_{\ell_1})$ by mapping $\alpha = (\alpha_1, \dots, \alpha_{\ell_1}) \in \{0, 1\}^{\ell_1}$ to $\sum_{i=1}^{\ell_1} \alpha_i \zeta_i$. Similarly map the last ℓ_2 input bits of D into $H_2 = \text{span}(\zeta_{\ell_1+1}, \dots, \zeta_\ell)$ by mapping $\beta = (\beta_1, \dots, \beta_{\ell_2})$ to $\sum_{i=1}^{\ell_2} \beta_i \zeta_{\ell_1+i}$. Let $H' = \text{span}(\zeta_1, \dots, \zeta_\ell)$ be the direct sum of H_1, H_2 and let $H = \text{span}(\zeta_1, \dots, \zeta_{\ell+1})$ noticing H' is subspace of codimension 1 inside H. (Recall $N > \ell$ so H is well-defined.) From here on we overload notation and treat α appearing in Theorem 2.2 as an element of H_1. Likewise β is an element of H_2 and $S_i(\beta)$ will be viewed as an element of H'. All arithmetic operations appearing henceforth are in $\mathbb{F}$.

Preprocessing variable selection functions For each $i \in \{1, 2, 3\}$ and $\beta \in H_1$ let

$$V_{i,\beta}(X) \triangleq X + S_i(\beta), \tag{2}$$

noticing it is a degree-1 polynomial. We also need the range-checking indexing function

$$V_{\text{range}}(X) \triangleq X + \zeta_{\ell+1} \tag{3}$$

We set the index set of variables Y and addressing functions V to be

$$J = \left\{(i, \beta) \mid i \in \{1, 2, 3\}, \beta \in \{0, 1\}^{\ell_2}\right\} \cup \{\text{range}\}.$$

The proof of the next claim follows by inspection of (2), (3).

Claim 5.3 (Preprocessing)**.** *For every $\alpha \in H_1$ and $\beta \in H_2$ we have $V_{i,\beta}(\alpha+\beta) = \alpha+\beta+S_i(\beta)$. In particular, all preprocessing functions appearing in (2) and (3) are projections.*

Sub-polynomials of Q The polynomial Q will be composed of $\text{poly}(\log T, n)$ many polynomials, each of constant degree and with a constant number of variables. We now define these sub-polynomials. For $\beta \in H_2$ let $b_1(\beta), b_2(\beta), b_3(\beta)$ denote the three sign-bits of the clause depending on β. Let $Q_\beta(Y_1, Y_2, Y_3)$ be the constraint that arithmetizes the relevant clause,

$$Q_\beta(Y_1, Y_2, Y_3) \triangleq (Y_1 - b_1(\beta)) \cdot (Y_2 - b_2(\beta)) \cdot (Y_3 - b_3(\beta))$$

Additionally, we need the range-testing polynomial $Q_{\text{range}}(Z) = Z^2 - Z$ whose roots are $\{0, 1\}$, i.e., $Q_{\text{range}}(Z) = \mathsf{Zero}_{\{0,1\}}(Z)$.

Gluing the subpolynomials to obtain Q To "glue" together the sub-polynomials discussed above we use selector polynomials, each selecting an affine subspace of H. These subspaces are defined for each $\beta \in H_2$:

$$\beta + H_1 = \{\alpha + \beta \mid \alpha \in H_1\}$$

Notice that $\mathsf{Sel}_{\beta+H_1 \subset H}$ is a selector polynomial for an affine subspace of H of codimension $O(\log \log T + \log n)$ in H, thus can be computed by a small circuit (cf. Lemma 5.2). Similarly let $\zeta_{\ell+1} + H_1$ denote the affine shift of H_1 by $\zeta_{\ell+1}$ and notice this is a space of constant codimension inside H. We now define

$$\begin{aligned}\hat{Q}(X, \mathcal{Y}) = \hat{Q}\left(X, \mathcal{Y} = \{Y_{i,\beta,a} \mid i \in \{1,2,3\}, \beta \in H_2, a \in \mathbb{F}_2\} \cup \{Y_{\text{range}}\}\right) = \\ \sum_{\beta \in H_2} \mathsf{Sel}_{\beta+H_1 \subset H}(X) \cdot Q_\beta(Y_{1,\beta}, Y_{2,\beta}, Y_{3,\beta}) + \mathsf{Sel}_{\zeta_{\ell+1}+H' \subset H}(X) \cdot Q_{\text{range}}(Y_{\text{range}})\end{aligned} \tag{4}$$

and

$$Q(X, Z, \mathcal{Y}) = \hat{Q}(X, \mathcal{Y}) - Z \cdot \mathsf{Zero}_H(X) \tag{5}$$

which conforms with (1).

Completeness and Soundness Follow from the next two claims.

Claim 5.4. *Let $A(X)$ be a polynomial. There exists a polynomial $B(X)$ such that (A, B) satisfy Q iff the polynomial*

$$Q_A(X) = \hat{Q}(X, \{Y_j \leftarrow A(V_j(X)) \mid j \in J\})$$

vanishes for every $h \in H$, i.e., $Q_A(h) = 0$ for every $h \in H$.

Proof. Recall that (A, B) satisfy Q iff

$$Q(X, Z \leftarrow B(X), \{Y_j \leftarrow A(V_j(X)) \mid j \in J\}) \equiv 0$$

Which means

$$Q_A(X) \equiv \mathsf{Zero}_H(X) \cdot B(X)$$

where equivalence is in the ring $\mathbb{F}[X]$. One direction is immediate: the right hand side in the last equation above vanishes on H by construction of Zero_H and hence $Q_A(X)$ must vanish on every $h \in H$ too. In the other direction, if $Q_A(X)$ vanishes on $h \in H$ then it is divisible by the polynomial $(X - h)$. This is because $\mathbb{F}[X]$ is a unique factorization domain. Hence $Q_A(X)$ is divisible by $\prod_{h \in H}(X - h) = \mathsf{Zero}_H(X)$, which means there exists $B(X)$ such that $Q_A(X) \equiv \mathsf{Zero}_H(X) \cdot B(X)$. □

Claim 5.5. *$Q_A(X)$ vanishes on every $h \in H$ if and only if:*

1. *$A(\alpha + \beta) \in \mathbb{F}_2$ for every $\alpha \in H_1, \beta \in H_2$, and (recalling $H' = H_1 + H_2$ is identified with $\{0,1\}^\ell$ and $\mathbb{F}_2$ is identified with $\{0,1\}$)*
2. *the boolean assignment $A : \{0,1\}^\ell \to \{0,1\}$ satisfies the succinct CSP defined by D.*

Proof. The first observation is that for any $h \in H$ there is at most one summand of (4) that does not vanish. This follows from the definition of the selector polynomial and because the sets $\beta + H_1$ and $\zeta_{\ell+1} + H'$ are all mutually disjoint.

To prove part 1 we start with the very last term in (4) and notice that, for $x \in H$, the selector polynomial selects only x of the form $x' + \zeta_{\ell+1}$ for $x' \in H'$. The function V_{range} adds $\zeta_{\ell+1}$ to its input, thus $(Y_{\text{range}} \leftarrow V_{\text{range}})$ evaluated on $x' + \zeta_{\ell+1}$ is simply x'. Hence $Q_A(X)$ vanishes on $x \in H' + \zeta_{\ell+1}$ iff $Q_{\text{range}}(A(\alpha+\beta)) = 0$ for every $\alpha \in H_1, \beta \in H_2$ which establishes part 1.

Part 2 follows by inspection, using the same rationale as used in proving part 1. Notice that the selector for $\beta + H_1$ evaluates to 1 on $x \in H$ if and only if x is of the form $\alpha + \beta$ for $\alpha \in H_1$. This, together with Claim 5.3, means that $(Y_{i,\beta} \leftarrow V_{i,\beta})$ evaluated on x is simply $\alpha + \beta + S_i(\beta)$. It follows from the definition of Q_β that the term corresponding to x vanishes iff the xth clause is satisfied by the assignment given by A to its three variables. This completes the proof. □

Computational Efficiency Postprocessing complexity is equal to the size of Q in (5), this size is poly(n), as implied by Lemma 5.2. Regarding preprocessing, it can be done via projections as shown in Claim 5.3. □

6 Preprocessing Complexity of the RS-PCPP verifier

In this section we prove Theorem 4.6. The completeness, soundness, query complexity, randomness and decision time of the RS-PCPP verifier are already given by [BS08, BGH$^+$05] (cf. [BCGT13b]) and here we focus solely on showing that preprocessing can be done via projections. This will be argued by following the proof in [BS08, Section 6] and specifying the query-format to the PCPP oracle and the preprocessing done to the randomness by the PCPP-verifier. We assume the reader is familiar with that proof (including the notation there) and point out, using footnotes, the minor modifications needed for our proof.

We start by recalling the RS-PCPP from [BS08, Section 6]. We consider a system for verifying proximity to $\text{RS}[\mathbb{F}, L, 2^{-a}]$ where $\mathbb{F}$ is a field of characteristic 2 and size $> 2^{k+a}$ and

L is an affine space[1] of dimension k. The proximity-verification problem for $\mathrm{RS}[\mathbb{F}, L, 2^{-a}]$ is reduced to $2 \cdot 2^{\lceil k/2 \rceil}$ many instances of the proximity-verification problems for $\mathrm{RS}[\mathbb{F}, L', 2^{-a}]$ where L' is an affine space of dimension $\leq \lceil k/2 \rceil + a$.[2] Applying the same kind of reduction recursively — we stop it when $\dim(L') \leq 2a$ — leads after $O(\log k)$ many iterations to constant-size domains L', for which the proximity-verification problem is solved by querying the relevant function on all of its domain and checking it has fractional degree less than 2^{-a}. Consequently, the combination of p and its RS-PCPP oracle can be viewed as residing in a tree. Each node v of the tree refers to a function $f_v : L_v \to \mathbb{F}$ where L_v is an affine space of dimension $k = k_v$, which we call also the dimension of v. The function p refers to the root of the tree. L_v is specified by a shift b_0 and a basis $b_1, \ldots, b_k$, so a query to f_v is specified by $(r_1, \ldots, r_k) \in \mathbb{F}_2^k$ and answered by $f(b_0 + \sum_i r_i b_i)$. One thing that complicates the situation is that the various functions residing in the tree must be *consistent*, i.e., they must give the same value to certain pairs of points. (This is because the functions f_u residing in the sons of v interpolate certain values of f_v, in a way explained later on.) We deal with this by redirecting queries from f_u to the relevant $f_{u'}$. In particular, our query-generating preprocessing algorithm runs in two phases: In the *down phase* we use randomness to go down the tree and reach a leaf, generating an initial set of queries. In the *up phase* we take our query and, if needed, redirect it either to a sibling or to its father, and continue the process. Thus, to prove Theorem 4.6 we will argue that each step of each of the phases is a projection. To state the two claims we explain more the structure of the RS-PCPP proof tree.

6.1 Overview of a single level of the RS-PCPP system

We continue to assume familiarity with [BS08, Section 6] and use the notation there. Assume $p : L \to \mathbb{F}$ belongs to $\mathrm{RS}[\mathbb{F}, L, 2^{-a}]$ and let $P(X)$ be the polynomial whose evaluation is p (i.e., $p(x) = P(x)$ for all $x \in L$ and $\deg(P) < |L|/2^a$). Let $\ell = \lfloor k/2 \rfloor$ and as in [BS08, Equation (6.2)] let

$$\begin{aligned} L_0 &\triangleq \mathrm{span}(b_1, \ldots, b_\ell), \quad \dim(L_0) = \ell \\ L_0' &\triangleq c_0 + L_0 + \mathrm{span}(c_1, \ldots, c_{a-1}), \quad \dim(L_0') = \ell + a - 1 \\ L_1 &\triangleq b_0 + \mathrm{span}(b_{\ell+1}, \ldots, b_k), \quad \dim(L_1) = k - \ell \end{aligned} \tag{6}$$

and $q(X) \triangleq q_{L_0}(x)$ be the subspace polynomial vanishing on $\mathrm{span}(b_1, \ldots, b_\ell)$. Notice $L_0' \cap L_1 = \emptyset$ so that we have a simpler version of [BS08, Equation (6.3)]: For $\beta \in L_1$ let

$$L_\beta \triangleq L_0' \cup \beta + L_0 + \mathrm{span}(c_1, \ldots, c_{a-1}), \quad \dim(L_\beta) = \ell + a \tag{7}$$

and notice $L_\beta \supset \beta + L_0$.

[1] [BS08, Section 6] assumes L is a linear space. The generalization to affine-spaces is immediate, cf. [BCGT13b].

[2] In [BS08, Section 6] $a = 3$, i.e., fractional degree is 1/8. This can be generalized to arbitrary constant a as we assume here.

By [BS08, Proposition 6.3] there exists a unique bivariate polynomial $Q(X,Y)$ satisfying $\deg_X(Q) < 2^\ell, \deg_Y(Q) < 2^{(k-\ell)-a}$ and

$$Q(X,Y) \equiv P(X) \mod Y - q(X), \quad \text{or equivalently} \quad P(X) = Q(X, q(X)). \tag{8}$$

Assuming p is the evaluation of P and using q and Q above, view $p : L \to \mathbb{F}$ as a partial bivariate function $p : T \to \mathbb{F}$ for $T \subset \mathbb{F} \times \mathbb{F}$

$$T = \bigcup_{\beta \in L_1} \{(\beta + L_0) \times \{q(\beta)\}\}. \tag{9}$$

In what follows for $T \subset \mathbb{F} \times \mathbb{F}$ let the β-row of T be $R_\beta[T] = \{\alpha \mid (\alpha, \beta) \in T\}$ and the α-column is $C_\alpha[T] = \{\beta \mid (\alpha, \beta) \in T\}$.

Next, as in [BS08, Definition 6.5] let $f : S \to \mathbb{F}$ where

$$S = \left(\bigcup_{\beta \in L_1} \{L_\beta \times \{q(\beta)\}\} \right) \setminus T$$

is supposedly the evaluation of Q on S and in what follows we denote by $\hat{f} : S \cup T \to \mathbb{F}$ the function that agrees with p on T and with f on S. Assuming p is the function residing at the root v, there is one row-edge leaving v for each nonempty row of $S \cup T$ and one column-edge leaving v for every α-column of S, where $\alpha \in L_0'$. The number and structure of these outgoing edges is given by the following analog of [BS08, Proposition 6.6].

Proposition 6.1 (Structure of S and T)**.** *The set $S \cup T$ is the disjoint union of $q(\beta)$-rows for $\beta \in L_1$. The $q(\beta)$-row of $S \cup T$ is the affine space L_β. Similarly, for every $\alpha \in L_0'$, the α-column of $S \cup T$ is the affine space $q(L_1)$, and it has no intersection with T.*

If $\hat{f}$ is indeed the evaluation of Q, then every row of $\hat{f}$ is a polynomial of degree $< 2^\ell$ and every column is of degree $< 2^{(k-\ell)-a}$. This leads to the following recursive definition of the RS-PCPP tree, which complements [BS08, Definition 6.5] by providing a more explicit query-access to nodes in the tree.

Definition 6.2 (RS-PCPP tree node labels)**.** *Let v be a node in a RS-PCPP tree, and assume it is labeled by affine shift b_0, basis $b_1, \ldots, b_k$, auxiliary shift c_0 and auxiliary basis $c_1, \ldots, c_{a-1}$ where $b_0, \ldots, b_k, c_0, \ldots, c_{a-1}$ are linearly independent (in particular, b_0 and c_0 are nonzero). A query to the node is specified by $r_1, \ldots, r_k$ and the answer is interpreted as $f(b_0 + \sum_{i=1}^k r_i b_i)$. (Notice that the auxiliary shift and basis are not used in querying f_v, rather they are needed for the recursion specified next.)*

If $k \le 2a$ then v is a leaf. Otherwise, let L_0, L_0', L_1, L_β be as in (6) and (7). Then v has the following edges

1. *v has $2^{k-\ell}$ row-edges, each edge labeled by $\beta \in L_1$, and the node u at the other end of the edge is labeled by the affine shift β and basis $b_1, \ldots, b_\ell, c_1, \ldots, c_{a-1}, (c_0 + \beta)$, auxiliary shift c_0' and auxiliary basis $c_1', \ldots, c_{a-1}'$ that are linearly independent of $\beta, b_1, \ldots, b_\ell, (c_0 + \beta), c_1, \ldots, c_{a-1}$.*

2. *Additionally, v has $2^{\ell+a-1}$ column-edges, each edge labeled by $\alpha \in L'_0$. The node u at the other end of the edge is labeled by shift $q(b_0)$ and basis $q(b_{\ell+1}), \ldots, q(b_k)$, auxiliary shift c'_0 and auxiliary basis $c'_1, \ldots, c'_{a-1}$ that are linearly independent of $q(b_0), q(b_{\ell+1}), \ldots, q(b_k)$.*

Intuitively, we assume the prover provides a function $f_v : L_v \to \mathbb{F}$ for each node v in our tree. We associate L_v with $T_v \subset \mathbb{F} \times \mathbb{F}$ as in (9), view f_v as a partial bivariate function with domain T_v. We then define a set S_v as in Proposition 6.1 and assume the prover provides for each $q(\beta)$-row of $S_v \cup T_v$ the evaluation of $\hat{f}$ restricted to that row. This evaluation resides in the node u at the end of the row-edge labeled by β. Similarly, the evaluation of $\hat{f}$ restricted to an α-column of $S_v \cup T_v$ resides in the node w at the end of the column-edge labeled by α. By definition, the $q(\beta)$-row of $S_v \cup T_v$ — which resides in node u — has non-empty intersection with T_v. On inputs in T_v we should require f_v to equal f_u. Similarly, every element in the α-column — which resides in node w — is also an element of some $q(\beta)$-row, so we require f_w to equal f_u for the relevant u. To enforce this consistency between functions residing in different nodes of the RS-PCPP tree, we define a query generation preprocess that traverses the RS-PCPP proof tree in two phases: In the *down phase* we use randomness to go down the tree and reach a leaf, generating an initial set of queries. In the *up phase* we take each query that was generated at the end of the down phase and, if needed, redirect it as follows: If the query is to a column-node (i.e., w reached from v via a column edge) then we refer the query to the proper row-sibling; if the query is to a row-node u (reached from v) and furthermore resides in T_v we refer it to the father v of u. Thus, to prove Theorem 4.6 we will argue that each step of each of the two phases is a projection. We start by describing the down phase.

Claim 6.3 (Down phase)**.** *At non-leaf v of dimension $k = k_v$ denote the k bits of initial randomness by $r = (r_1, \ldots, r_k)$. The down step consists of*

- *Partitioning the bits into $r'' = (r_1, \ldots, r_\ell)$ and $r''' = (r_{\ell+1}, \ldots, r_k)$*
- *Appending a constant number $a - 1$ of new bits $r' = (r'_1, \ldots, r'_{a-1})$ to r''.*
- *Tossing a coin r'_0 and based on its outcome either use r'' as a label for a column-edge $v \to u$ and apply another down-step to u with randomness r''', or use r''' as a label for a row-edge $v \to u'$ and apply another down-step to u' with randomness r''.*

This down phase process is initialized at the root with k bits of randomness, and continued until a leaf u of dimension $k \leq 2a$ is reached. (Recall a is a constant.) All 2^a elements of f_v are designated as queries and we apply the up phase to each one of them separately. The up phase consists of $O(\log k)$ many steps, at most 2 up-steps at each level of the tree.

Claim 6.4 (Up phase)**.** *Let u be a node of dimension k_u and let $r = (r_1, \ldots, r_k)$ be the randomness used to query it. If u is the root, then the up-phase terminates and the query r is sent to f_u. Otherwise, let v be the father of u and let r' be the bit-sequence specifying its edge-label. A number a of the bits of r' are examined in a non-adaptive manner and based on their value, the up-step chooses one of three possibilities: (i) make query r to f_u, (ii) redirect the query r' to the son u' of v with edge $v \to u'$ labeled r, or (iii) redirect the query $r \circ r'$ to v.*

Proof of Theorem 4.6. Proof length is proved in [BS08]. As shown there, a single invocation of the RS-PCPP verifier has query complexity $O(1)$ (over alphabet $\mathbb{F}$) and soundness $1/k^c$ for a universal constant c.

We now argue that preprocessing can be achieved by a projection. First, notice that generating $R+y$ from R is a projection according to the last bullet of Theorem 1.1. Having fixed the initial randomness, the key point is that only $O(\log k)$ bits are examined in all down-steps and up-steps. Hence, consider a modified verifier which fixes in advance every possible setting for these $O(\log k)$ bits. The query complexity of the modified verifier is poly(k). Soundness can only increase because of this modification. Crucially, once these $O(\log k)$ bits are fixed in advance, the remaining preprocessing is a projection according to the last bullet of Theorem 1.1. A down-phase consists of permuting the bits; at the leaf a constant number of bits are flipped; and the up phase consists yet again of permuting the bits. This completes the proof of Theorem 4.6. □

6.2 Down phase — Proof of Claim 6.3

Assume $f : L \to \mathbb{F}$ resides at node v; we have k bits of randomness $r_1, \ldots, r_k$ passed on from v's father and a new bits of randomness $r'_0, \ldots, r'_a$; L is specified by shift b_0 and basis $b_1, \ldots, b_k$ for L, and v has an auxiliary shift c_0 and auxiliary basis $c_1, \ldots, c_{a-1}$ for. Then by Definition 6.2 v has one row-son labeled by β for each $\beta \in L_1$, and one column-son labeled by α for each $\alpha \in L'_0$. We use the first bit of new randomness r'_0 to select a row ($r'_0 = 0$) or column ($r'_0 = 1$) edge. If a row-edge is selected we use $r_{\ell+1}, \ldots, r_k$ to select a $\beta \in L_1$ (recall from (6) that $\dim(L_1) = k - \ell$) and pass $(r_1, \ldots, r_\ell, r'_1, \ldots, r'_a)$ as randomness to the row-node u_β. (Definition 6.2 and (7) imply $\dim(u_\beta) = \ell + a$.) If a column-edge is selected we use $(r_1, \ldots, r_\ell, r'_1, \ldots, r'_{a-1})$ to select a column $\alpha \in L'_0$ (recall from (6) that $\dim(L'_0) = \ell + a - 1$) and pass $(r_{\ell+1}, \ldots, r_k)$ to the column-node u_α of dimension $k - \ell$.

Notice that in either case (row/column) the operations performed on $r_1, \ldots, r_k, r'_0, \ldots, r'_a$ consist of *(i)* partitioning the bits in a fixed way into a single bit (r'_0) and two roughly equisized sets R_0, R_1. Then, based on r'_0 we either use R_0 as an edge label and pass R_1 as randomness to the next node, or vice versa (use R_1 as and edge label and pass R_0 as randomness to next node). Thus, each down phase examines $O(1)$ bits and then (possibly) a fixed permutation of the remaining bits. This completes the proof of Claim 6.3.

6.3 Up phase — Proof of Claim 6.4

If u is the root, then query f_u at $(r_1, \ldots, r_k)$. Otherwise, there are two cases to consider:

Column nodes In this case suppose u is an $(k-\ell)$-dimensional column node whose father is an k-dimensional node v and the edge leading from v to u is labeled by $\alpha \in L'_0$ and the query to f_u is $r_1, \ldots, r_{k-\ell}$. The crucial point is that the shift and basis for querying f_u are $q(b_0)$ and $q(b_1), \ldots, q(b_\ell)$, so the query to f_u is $q(\beta)$ where $\beta = b_0 + \sum_{i=1}^{k-\ell} r_i q(b_i)$. On the other hand, the shift for the row-sibling u' of u down the edge labeled by $r_1, \ldots, r_{k-\ell}$ is β and the basis is $b_1, \ldots, b)\ell, c_1, \ldots, c_{a-1}, (c_0 + \beta)$. Thus, assuming f_u is the restriction of $\hat{f}$ to the α-column, and $f_{u'}$ is the restriction of $\hat{f}$ to the $q(\beta)$-row, we see that the answer of f_u at α should equal the answer of $f_{u'}$ on query β and this is precisely case *(ii)* of Claim 6.4.

Row nodes In this case suppose u is a $\ell + a$-dimensional row-node whose father is an k dimensional node v and the edge leading from v to u is labeled by β and the query to f_u is labeled by $\alpha = \beta+\sum_{i=1}^{\ell} r_i b_i+\sum_{j=1}^{a-1} r_{\ell+j} c_j + r_{\ell+a}(c_0+\beta)$ where $r_1, \ldots, r_{\ell+a}$ is the randomness used to specify the query. We need to redirect the query to f_v if and only if $\alpha \in L_\beta \cap T_v$ which, inspection reveals, happens iff $r_{\ell+a} = 1$ and $r_{\ell+1} = \ldots = r_{\ell+a-1} = 0$. This can be determined by inspecting a bits of the randomness, and if needed, the query comprised of randomness $r_1, \ldots, r_{\ell-k}$ is concatenated with the ℓ bits of randomness specifying the edge β and the total k bits are redirected to the father node v. This completes case *(iii)* of Claim 6.4.

We conclude that in each case, the up-phase step occurring at level i of the tree involves at most 2 hops — one from column-node to a sibling row-node, the other from a row-node to its father. In each case the processing needed to be done involves inspecting at most a bits, and based on this redirecting the query by possibly permuting its randomness bits. This completes the proof of Claim 6.4.

Acknowledgements

The first co-author is grateful to the SCIPR-lab (www.scipr-lab.org) members — Alessandro Chiesa, Daniel Genkin, Lior Greenblatt, Shaul Kfir, Michael Riabzev, Gil Timnat, Eran Tromer and Madars Virza — for helpful discussions. The second co-author thanks Ramamohan Paturi for a conversation on MAX3SAT.

References

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[ASE92] Noga Alon, Joel H. Spencer, and Paul Erdős. *The Probabilistic Method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc., 1992.

[BCGT13a] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *ACM Innovations in Theoretical Computer Science conf. (ITCS)*, pages 401–414, 2013.

[BCGT13b] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *ACM Symp. on the Theory of Computing (STOC)*, pages 585–594, 2013.

[BGH$^+$05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short PCPs verifiable in polylogarithmic time. In *IEEE Conf. on Computational Complexity (CCC)*, pages 120–134, 2005.

[Blu84] Norbert Blum. A boolean function requiring 3n network size. *Theoretical Computer Science*, 28:337–345, 1984.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. on Computing*, 38(2):551–607, 2008.

[BSGH$^+$06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. on Computing*, 36(4):889–974, 2006.

[Coo73] Stephen A. Cook. A hierarchy for nondeterministic time complexity. *J. of Computer and System Sciences*, 7(4):343–353, 1973.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. of the ACM*, 54(3):12, 2007.

[DK11] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In *Symp. on Math. Foundations of Computer Science (MFCS)*, pages 256–265, 2011.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. on Computing*, 36(4):975–1024, 2006.

[FSUV13] Bill Fefferman, Ronen Shaltiel, Christopher Umans, and Emanuele Viola. On beating the hybrid argument. *Theory of Computing*, 9:809–843, 2013.

[GMR13] Parikshit Gopalan, Raghu Meka, and Omer Reingold. DNF sparsification and a faster deterministic counting algorithm. *Computational Complexity*, 22(2):275–310, 2013.

[Her11] Timon Hertli. 3-SAT faster and simpler - unique-SAT bounds for PPSZ hold in general. In *IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 277–284, 2011.

[IKW01] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *IEEE Conf. on Computational Complexity (CCC)*, 2001.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. of Computer and System Sciences*, 65(4):672–694, 2002.

[JMV13] Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. Available at http://www.ccs.neu.edu/home/viola/, 2013.

[KL80] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *ACM Symp. on the Theory of Computing (STOC)*, pages 302–309, 1980.

[LVW93] Michael Luby, Boban Veličković, and Avi Wigderson. Deterministic approximate counting of depth-2 circuits. In *2nd Israeli Symposium on Theoretical Computer Science (ISTCS)*, pages 18–24, 1993.

[Mie09] Thilo Mie. Short pcpps verifiable in polylogarithmic time with $o(1)$ queries. *Ann. Math. Artif. Intell.*, 56(3-4):313–338, 2009.

[MTY11] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing HSSW algorithm for 3-SAT. *CoRR*, abs/1102.3766, 2011.

[NEU12] NEU. From RAM to SAT. Available at http://www.ccs.neu.edu/home/viola/, 2012.

[NN90] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. In *22nd ACM Symp. on the Theory of Computing (STOC)*, pages 213–223. ACM, 1990.

[Oli13] Igor C. Oliveira. Algorithms versus circuit lower bounds. *CoRR*, abs/1309.0249, 2013.

[SFM78] Joel I. Seiferas, Michael J. Fischer, and Albert R. Meyer. Separating nondeterministic time complexity classes. *J. of the ACM*, 25(1):146–167, 1978.

[SW13] Rahul Santhanam and Ryan Williams. On medium-uniformity and circuit lower bounds. In *IEEE Conf. on Computational Complexity (CCC)*, 2013.

[Vio07] Emanuele Viola. Pseudorandom bits for constant-depth circuits with few arbitrary symmetric gates. *SIAM J. on Computing*, 36(5):1387–1403, 2007.

[Vio13] Emanuele Viola. Challenges in computational lower bounds. Available at http://www.ccs.neu.edu/home/viola/, 2013.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *42nd ACM Symp. on the Theory of Computing (STOC)*, pages 231–240, 2010.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *IEEE Conf. on Computational Complexity (CCC)*, pages 115–125, 2011.

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. on Computing*, 42(3):1218–1244, 2013.

[Wil14] Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. *CoRR*, abs/1401.2444, 2014.

[Zák83] Stanislav Zák. A turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de