Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 157-168
www.stacs-conf.org

FINDING IRREFUTABLE CERTIFICATES FOR S_2^p VIA ARTHUR AND MERLIN

VENKATESAN T. CHAKARAVARTHY AND SAMBUDDHA ROY

IBM India Research Lab, New Delhi.
E-mail address: {vechakra,sambuddha}@in.ibm.com

Abstract. We show that $S_2^p \subseteq P^{prAM}$, where S_2^p is the symmetric alternation class and prAM refers to the promise version of the Arthur-Merlin class AM. This is derived as a consequence of our main result that presents an FP^{prAM} algorithm for finding a small set of "collectively irrefutable certificates" of a given S_2-type matrix. The main result also yields some new consequences of the hypothesis that NP has polynomial size circuits. It is known that the above hypothesis implies a collapse of the polynomial time hierarchy (PH) to $S_2^p \subseteq ZPP^{NP}$ [5, 14]. Under the same hypothesis, we show that PH collapses to P^{prMA}. We also describe an FP^{prMA} algorithm for learning polynomial size circuits for SAT, assuming such circuits exist. For the same problem, the previously best known result was a ZPP^{NP} algorithm [4].

1. Introduction

We consider the problem of finding irrefutable certificates for the symmetric alternation class S_2^p. The class S_2^p was introduced by Russell and Sundaram [17] and independently, by Canetti [6]. A language L in the class S_2^p is characterized by an interactive proof system of the following type. The proof system consists of two computationally all-powerful provers called the Yes-prover and the No-prover, and a polynomial time verifier. The verifier interacts with the two provers to ascertain whether or not an input string x belongs to the language L. The Yes-prover and the No-prover make contradictory claims: $x \in L$ and $x \notin L$, respectively. Of course, only one of them is honest. To substantiate their claims, the provers provide strings y and z as certificates. The verifier analyzes the input x and the two certificates and votes in favor of one of the provers. If the Yes-prover wins the vote, we say that y *beats* z and we say that z *beats* y, otherwise. The requirement is that, if $x \in L$, then the Yes-prover must have a certificate y that beats any certificate z given by the No-prover. Similarly, if $x \notin L$, the No-prover must have a certificate z that beats any certificate y given by the Yes-prover. We call certificates satisfying the above requirements as *irrefutable certificates* (written IC). Clearly, for any input string, only the honest prover has an IC.

Cai [5] showed that $S_2^p \subseteq ZPP^{NP}$. Let us rephrase this result: for any language $L \in S_2^p$, we have a ZPP^{NP} algorithm that takes an input string and decides whether the Yes-prover

Key words and phrases: Symmetric alternation, promise-AM, Karp–Lipton theorem, learning circuits.

has an IC or the NO-PROVER has an IC. The main purpose of this paper is to study the problem of *finding* IC's for an input string.

The above problem and the related issues regarding S_2^p can conveniently be described in terms of Boolean matrices. Let L be a language in S_2^p and x be an input string. Let n and m, denote the length of the certificates of the YES-PROVER and NO-PROVER, respectively. We model the behaviour of the verifier on the input x in the form of a $2^n \times 2^m$ Boolean matrix M. In the matrix M, the rows correspond to the certificates of the YES-PROVER and the columns correspond to the certificates of the NO-PROVER. For certificates $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$, if y beats z, then we set $M[y,z] = 1$ and if z beats y, then we set $M[y,z] = 0$. Notice that the matrix M has either a row full of 1's or a column full of 0's. The first scenario happens, when $x \in L$ (here, the row full of 1's corresponds to an IC of the YES-PROVER). Similarly, the second scenario happens, when $x \notin L$ (here, the column full of 0's corresponds to an IC of the NO-PROVER). We call any Boolean matrix satisfying the above condition as an S_2-*type* matrix. A row full of 1's is called a row-side IC and a column full of 0's is called a column-side IC. (Notice that a Boolean matrix cannot have both.) Though the matrix M is exponentially large in the size of the input $|x|$, it can be *succinctly encoded* in the form of a Boolean circuit C having size polynomial in $|x|$. The circuit C takes as input $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$ and outputs $C(y,z) = M[y,z]$. The circuit achieves this by simulating the verifier's algorithm on the input x. Using standard techniques, we can construct the desired circuit C in time polynomial in $|x|$.

Problems regarding S_2^p can now be expressed as problems on S_2-type matrices, presented succinctly in the form of circuits. First, let us consider the basic problem of membership testing for a language $L \in S_2^p$: given a string x, determine whether $x \in L$ or not. This is equivalent to following problem on S_2-type matrices.

MEMBERSHIP TESTING. Given an S_2-type matrix M, presented succinctly in the form of a circuit, distinguish between the two cases: (i) there exists a row-side IC; (ii) there exists a column-side IC.

Cai [5] showed that $S_2^p \subseteq ZPP^{NP}$. Equivalently, this result presents a ZPP^{NP} algorithm for the MEMBERSHIP TESTING problem. We consider the more general problem of *finding* an IC for a given S_2-type matrix.

Problem FINDIC: Given an S_2-type matrix M, presented succinctly in the form of a circuit, output an IC either on the row side or on the column side.

Via a simple observation, we show that if there exists a ZPP^{NP} algorithm for the FINDIC problem, then the polynomial time hierarchy (PH) collapses. In summary, we can determine in ZPP^{NP} whether an IC is found among the rows or among the columns; but, we cannot find an IC in ZPP^{NP}, unless PH collapses. So, we study the easier problem of finding a set of *collectively irrefutable certificates* (written CIC).

We say that a set of rows Y *collectively beats* a column z, if some row $y \in Y$ beats z. The set Y is said to be a *row-side* CIC, if Y collectively beats every z. The notion of column-side CIC is defined analogously. Notice that an arbitrary Boolean matrix may have both a row-side CIC and a column-side CIC. However, the existence of a row-side CIC precludes there being a column-side IC. Thus, in the case of S_2-type matrices, a row-side CIC shows that there exists a row-side IC (which in turn, means that the input string $x \in L$). Therefore, a row-side CIC is as useful as a row-side IC, in certifying that $x \in L$. Our main result provides an algorithm for finding a CIC of small size (logarithmic in the size of the input matrix).

Problem FINDCIC. Given an S_2-type matrix M of size $2^n \times 2^m$, presented succinctly in the form of a circuit, output either a row-side CIC or a column-side CIC of size $\max\{n, m\}$.

Our main result presents an FP^{prAM} algorithm for the FINDCICproblem, i.e., the algorithm runs in (deterministic) polynomial time making queries to an prAM oracle; prAM refers to the promise version of the Arthur-Merlin class AM.

Main Result. We present an FP^{prAM} algorithm for the FINDCIC problem.

We note that the problem FINDCIC can also be solved by a ZPP^{NP} algorithm; such an algorithm is implicit in the work of Cai [5] and Fortnow et al. [9]. The containment relationships between FP^{prAM} and ZPP^{NP} are not known. This issue is discussed in more detail below.

An immediate corollary of the main result is that $S_2^p \subseteq P^{prAM}$. This gives a nice counterpart to Cai's result [5] that $S_2^p \subseteq ZPP^{NP}$. The containment relationships between P^{prAM} and ZPP^{NP} are unknown. (In fact, it has been a long standing open problem to put AM in Σ_2^p). However, we can show that $P^{prAM} \subseteq BPP^{NP}$. Moreover, Cai's result can also be derived from the main result.

It is known that $P^{NP} \subseteq S_2^p$ [17] and one of the most challenging open problems regarding S_2^p asks whether S_2^p is contained in P^{NP}. Working under a larger framework, Shaltiel and Umans [19] also studied this issue and derived the result $S_2^p = P^{NP}$, under a suitable hardness hypothesis. This was achieved by derandomizing Cai's construction for $S_2^p \subseteq ZPP^{NP}$. The above-mentioned hardness hypothesis was the one used by Miltersen and Vinodchandran [15] to derandomize AM to get AM = NP: there exists a language L in $NE \cap coNE$ so that for all but finitely many n, $L \cap \{0,1\}^n$ has SV-nondeterministic circuit complexity at least $2^{\epsilon n}$. Thus, under the above hypothesis, Shaltiel and Umans showed that $S_2^p = P^{NP}$. Our claim that $S_2^p \subseteq P^{prAM}$ yields an alternative proof of the above result. This is obtained by appealing to the hitting set generator of Miltersen and Vinodchandran [15]. The details will be included in the full version of the paper.

The main result yields two new consequences of the assumption that NP has polynomial size circuits. Under the above assumption, Karp and Lipton [13] showed that the polynomial time hierarchy (PH) collapses to Σ_2^p. Subsequently, their result has been strengthened: Köbler and Watanabe [14] derived the collapse $PH = ZPP^{NP}$; Sengupta observed that $PH = S_2^p \subseteq ZPP^{NP}$ (see [5]); recently, the collapse was improved to $PH = O_2^p \subseteq S_2^p$ [7]. It has been a challenging open problem to get the collapse down to P^{NP}. We derive a weaker result: if NP has polynomial size circuits, then $PH = P^{prMA}$. It is worthwhile to compare this new collapse result with the earlier ones. Though it is known that $P^{MA} \subseteq S_2^p$ [17], it is not clear whether P^{prMA} is contained in S_2^p. However, we can show that $P^{prMA} \subseteq ZPP^{NP}$ (by extending the known result that $MA \subseteq ZPP^{NP}$ [1, 11]).

One implication of the new collapse result is that P^{prMA} cannot have $SIZE(n^k)$ circuits, for any fixed k. However, a stronger result is known: in a recent breakthrough, Santhanam [18] proved the above circuit lowerbound for the class prMA.

In the above context, our next result deals with the problem of learning polynomial size circuits for SAT. Under the assumption that NP has polynomial size circuits, Bshouty et al. [4] designed a ZPP^{NP} algorithm that finds a correct circuit for SAT at a given length. We improve their result by presenting a FP^{prMA} algorithm for the same task.

Finally, we show how to generalize our main result to the case of arbitrary Boolean matrices (that may not necessarily be of S_2-type). For this, we make use of a nice and interesting lemma by Goldreich and Wigderson [10]: they showed that *any* $2^n \times 2^m$ Boolean

matrix M contains a row-side CIC of size m or a column-side CIC of size n (or both). We consider the scenario where the matrix M is presented succinctly in the form of a circuit and describe an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding such a CIC; but, our algorithm suffers a small blow-up in the size of the output CIC. The algorithm finds a row-side CIC of size m^2 or a column-side CIC of size n^2.

For lack of space, the details of the above generalization and proofs for some of the results are omitted in this paper. These will be included in the full version of the paper.

Proof Techniques. The proof of our main result has a flavor similar to that of Cai's result [5]. The proof involves a variant of self-reduction and the tools of approximate counting and testing whether a set is "large" or "small". For the latter two tasks, we borrow ideas from the work of Jerrum et al. [12], Stockmeyer [21] and Sipser [20]. We put together all these ideas and show how to solve our problem using a prAM oracle. Our exposition is largely self-contained.

2. Preliminaries

In this section, we develop definitions and notations used throughout the paper.

Symmetric Alternation. A language L is said to be in the class S_2^p, if there exists a polynomial time computable Boolean predicate $V(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that for any x, we have

$$\begin{aligned} x \in L &\implies (\exists y \in \{0,1\}^n)(\forall z \in \{0,1\}^m)[V(x,y,z)=1], \text{and} \\ x \notin L &\implies (\exists z \in \{0,1\}^m)(\forall y \in \{0,1\}^n)[V(x,y,z)=0], \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. We refer to the y's and z's above as *certificates*. The predicate V is called the *verifier*.

Matrix representation of the verifier's computation. Let L be a language in S_2^p via a verifier predicate V. Fix an input string x. It is convenient to represent the behaviour of the verifier on various certificates in the form of a matrix. Define a Boolean $2^n \times 2^m$ matrix M, such that for $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$, $M[y,z] = V(x,y,z)$. Thus, any row or column in M corresponds to a certificate. We call M as the matrix corresponding to the input x. Matrices constructed in the above fashion have some special properties that are derived the from the definition of S_2^p.

S_2-type matrices and irrefutable certificates. Let M be a $2^n \times 2^m$ Boolean matrix. For a row $y \in \{0,1\}^n$ and a column $z \in \{0,1\}^m$, if $M[y,z] = 1$, then y is said to *beat* z; similarly, z is said to beat y, if $M[y,z] = 0$. A row y is called a *row-side* IC, if y beats every column $z \in \{0,1\}^m$; a column z is called a *column-side* IC if z beats every row $y \in \{0,1\}^m$. Notice that a matrix cannot have both a row-side IC and a column-side IC. The matrix M is said to be an S_2-type matrix, if it has either a row-side IC or a column-side IC. A set of rows Y is called a row-side CIC, if for every column z, there exists a row $y \in Y$ such that y beats z. Similarly, a set of columns Z is called a column-side CIC, if for every row y, there exists a column $z \in Z$ such that z beats y.

Remark. Let us put the above discussion in the context of a language $L \in \mathrm{S}_2^p$ and make some simple observations. For any input string x, the matrix M corresponding to x is an S_2-type matrix. The matrix M will have a row-side IC, if and only if $x \in L$; similarly, M will have a column-side IC, if and only if $x \notin L$.

Succinct encoding of matrices and sets. A Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is said to *succinctly encode* a Boolean $2^n \times 2^m$ matrix M, if for all $y \in \{0,1\}^n$ and

$z \in \{0,1\}^m$, we have $C(y,z) = M[y,z]$. A Boolean circuit $C : \{0,1\}^m \to \{0,1\}$ is said to *succinctly encode* a set $X \subseteq \{0,1\}^m$, if for all $x \in \{0,1\}^m$, $x \in X \iff C(x) = 1$.

Remark. Let L be a language in S_2^p via a verifier V. Let x be an input string with the corresponding matrix M. Using standard techniques, we can obtain a Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ such that $C(y,z) = V(x,y,z)$. Given the input x, the above task can be performed in time polynomial in $|x|$. The size of the circuit is also polynomial in $|x|$. Notice that the above circuit C succinctly encodes the matrix M.

Complexity classes. We use standard definitions for complexity classes such as P, NP, P/poly, MA, AM, $\mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{BPP}^{\mathrm{NP}}$ [8, 16]. Below, we present definitions for promise and function classes, that are central to our paper.

Promise languages. A promise language Π is a pair (Π_1, Π_2), where $\Pi_1, \Pi_2 \subseteq \Sigma^*$, such that $\Pi_1 \cap \Pi_2 = \emptyset$. The elements of Π_1 are called the *positive instances* and those of Π_2 are called the *negative instances*.

Promise MA (prMA). A promise language $\Pi = (\Pi_1, \Pi_2)$ is said to be in the promise class prMA, if there exists a polynomial time computable Boolean predicate $A(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that, for all x, we have

$$\begin{aligned} x \in \Pi_1 &\implies (\exists y \in \{0,1\}^n)(\forall z \in \{0,1\}^m)[A(x,y,z) = 1], \text{and} \\ x \in \Pi_2 &\implies (\forall y \in \{0,1\}^n) \Pr_{z \in \{0,1\}^m}[A(x,y,z) = 1] \leq \frac{1}{2}, \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. The predicate A is called *Arthur's predicate*.

Promise AM (prAM). A promise language $\Pi = (\Pi_1, \Pi_2)$ is said to be in the promise class prAM, if there exists a polynomial time computable Boolean predicate $A(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that, for all x, we have

$$\begin{aligned} x \in \Pi_1 &\implies (\forall y \in \{0,1\}^n)(\exists z \in \{0,1\}^m)[A(x,y,z) = 1], \text{and} \\ x \in \Pi_2 &\implies \Pr_{y \in \{0,1\}^n}[(\exists z \in \{0,1\}^m)A(x,y,z) = 1] \leq \frac{1}{2}, \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. The predicate A is called *Arthur's predicate*.

Oracle access to promise languages. Let A be an algorithm and $\Pi = (\Pi_1, \Pi_2)$ be a promise language. When the algorithm A asks a query q, the oracle behaves as follows: if $q \in \Pi_1$, the oracle replies "yes"; if $q \in \Pi_2$, the oracle replies "no"; if q is neither in Π_1 nor in Π_2, the oracle may reply "yes" or "no". We allow the algorithm to ask queries of the third type. The requirement is that the algorithm should be able to produce the correct answer, regardless of the answers given by the oracle to the queries of the third type.

Function classes. For a promise language Π, the notation FP^{Π} refers to the class of functions that are computable by a polynomial time machine, given oracle access to Π. For a promise class $\mathcal{C}$, we denote by $\mathrm{FP}^{\mathcal{C}}$, the union of FP^{Π}, for all $\Pi \in \mathcal{C}$. Regarding $\mathrm{ZPP}^{\mathrm{NP}}$, we slightly abuse the notation and use this to mean both the standard complexity class and the function class. The function class $\mathrm{ZPP}^{\mathrm{NP}}$ contains functions computable by a zero-error probabilistic polynomial time algorithm given oracle access to NP; the algorithm either outputs a correct value of the function or "?", the latter with a small probability.

3. Main Result: Finding Collectively Irrefutable Certificates

In this section, we study the problem of finding irrefutable certificates for S_2-type matrices. As discussed in the introduction, finding a single IC in ZPP^{NP} would collapse polynomial time hierarchy (PH).

Theorem 3.1. *If there exists a* ZPP^{NP} *algorithm for the* FINDIC *problem then* PH = BPP^{NP}.

Here, we focus on finding "small" CIC's. We present an FP^{prAM} algorithm for the FINDCIC problem.

Theorem 3.2. *There exists a polynomial time algorithm which solves the following problem, given oracle access to* prAM. *The algorithm takes as input a circuit C succinctly encoding a S_2-type matrix M of size $2^n \times 2^m$ and produces either a row-side* CIC *of size m or a column-side* CIC *of size n.*

For ease of exposition, we have divided the proof into multiple small steps; in each step, the given problem is reduced (in the Turing sense) to a simpler problem. The final algorithm is obtained by composing these reductions. The various steps are grouped into two phases. The first phase reduces the given problem to a problem that we call Prefix Ratio Goodness Testing (PRGT). The second phase describes an algorithm for PRGT.

3.1. Reduction to Prefix Ratio Goodness Testing

We are given an S_2-type matrix M. By definition, M is guaranteed to have either a row-side IC or a column-side IC. Our goal is to find a small CIC. This problem reduces to the problem addressed in Lemma 3.3, given below. The lemma presents an FP^{prAM} algorithm for finding a small row-side CIC for matrices that are guaranteed to have a row-side IC. Via an easy transformation, we can obtain an analogous algorithm for finding a small column side CIC for matrices guaranteed to have a column-side IC. We run both these algorithms on the given S_2-type matrix M. Notice that one of these runs must output a CIC. The other run would output some arbitrary result, because the input matrix does not satisfy the requirements of the concerned algorithm. We check which of the two outputs is indeed a CIC and output the same. This check can be performed by making a single NP query. Thus, we get the FP^{prAM} algorithm claimed in Theorem 3.2.

Lemma 3.3. *There exists an* FP^{prAM} *algorithm that takes as input a circuit C succinctly encoding a $2^n \times 2^m$ matrix M that is guaranteed to have a row-side* IC *and outputs a row-side* CIC *of size m.*

The algorithm computes the required CIC using a standard iterative approach: in each iteration, we find a row y that beats at least half of the columns that are as yet unbeaten by the rows found in the previous iterations. Formally, we start with an empty set Y and proceed iteratively, adding a row to Y in each iteration. Consider the k^{th} iteration. Let U_k be the set of columns as yet unbeaten by any row in Y (i.e., $U_k = \{z \in \{0,1\}^m |$ no $y \in Y$ beats $z\}$). We find a row y^* such that y^* beats at least half the columns in U_k and add y^* to Y. Notice that such a y^* exists, since we are guaranteed that M has a row-side IC. Clearly, the algorithm terminates in m steps and produces a row-side CIC of size m. Of course, the main step lies in finding the required y^* in each iteration. This task is accomplished by the algorithm described in Lemma 3.4, given below. The algorithm, in fact, solves a more general problem: given *any* set of columns $X \subseteq \{0,1\}^m$, it produces a

row beating at least half of the columns in X. In each iteration, we invoke the algorithm by setting $X = U_k$. There is one minor issue that needs to be addressed: the set U_k could be exponentially large. So, we represent the set U_k in the form of a circuit C' succinctly encoding it. For this, given any column $z \in \{0,1\}^m$, C' has to test whether z is beaten by any of the rows in Y. This test involves a simulation of $C(y,z)$, for all $y \in Y$. Since Y contains at most m rows, we can succinctly encode U_k by a circuit of size polynomial in the size of C. We have proved Lemma 3.3, modulo Lemma 3.4.

Lemma 3.4. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes two inputs: (i) a* $2^n \times 2^m$ *Boolean matrix* M *that is guaranteed to have a row-side* `IC`*; (ii) a set of columns* $X \subseteq \{0,1\}^m$*. It outputs a row* y^* *that beats at least half the columns in* X*. The matrix* M *and the set* X *are presented succinctly in the form of circuits.*

We build the required string y^* (of length n) in n iterations using an approach similar to self-reduction. We maintain a prefix of y^* and add one suitable bit in each iteration. However, we cannot directly employ self-reduction, since a query of the form "does there exist a row that beats at least half the columns in X" is a PP query and we cannot hope to find the answer using a prAM oracle. Nevertheless, we show how to converge on a y^* by performing self-reduction that incurs a small amount of "loss" in the "goodness" of the final y^*, in each iteration. We formalize the notion of goodness and then describe the algorithm.

Consider a $2^p \times 2^q$ Boolean matrix A and let $Q \subseteq \{0,1\}^q$ be a subset of the columns of A. For a row $y \in \{0,1\}^p$, define $\mu(y,Q)$ to be the fraction of columns in Q that y beats: $\mu(y,Q) = |\{z \in Q : y \text{ beats } z\}|/|Q|$. Let α be a string of length at most p. We say that a row $y \in \{0,1\}^p$ *extends* α, if α is a prefix of y. For $\rho \leq 1$, we say that α is ρ–good for Q, if there exists a row y extending α such that $\mu(y,Q) \geq \rho$.

The algorithm claimed in Lemma 3.4 constructs the string y^* in n iterations. Starting with the empty string, we keep building a prefix of y^*. At the end of the $(k-1)^{st}$ iteration, we have a prefix α_{k-1} of length $k-1$. In the k^{th} iteration, we extend α_{k-1} by one more bit b to get a prefix α_k of length k. To start with, we are guaranteed the existence of a row-side `IC` in M, meaning a row with goodness=1. Consider the k^{th} iteration. Suppose the prefix α_{k-1} is ρ–good with respect to X, for some ρ. Below, we describe a mechanism for finding a bit b such that the string $\alpha_{k-1}b$ is $(\rho-\epsilon)$–good. The value ϵ is a parameter to be fixed suitably later. Thus, in each iteration, we suffer a loss of ϵ and so, the accumulated loss at the end of n iterations is $n\epsilon$. Choosing ϵ suitably small, we get a string y^* having goodness at least $1/2$.

The main step lies in choosing a suitable bit b in each iteration. Consider the k^{th} iteration. At the end of $(k-1)st$ iteration, we have prefix α of length $k-1$. Write $\rho = 1-(k-1)\epsilon$. By induction, assume that α is ρ-good. Our task is to find a bit b such that αb is $(\rho-\epsilon)$-good. This is accomplished by invoking the algorithm given in Lemma 3.5, which solves the Prefix Ratio Goodness Testing problem (`PRGT`), defined below. The main observation is that at least one of $\alpha 0$ or $\alpha 1$ is ρ-good, because α is ρ-good. We run the algorithm given in Lemma 3.5 twice with $\beta = \alpha 0$ and $\beta = \alpha 1$ as inputs, respectively. By the above observation, at least one these two runs must output "yes". Let b be a bit such that the algorithm outputs "yes" on input αb. We choose b as the required bit. It is easy to see that αb is $(\rho-\epsilon)$-good; otherwise, the algorithm should have output "no" on αb.

Proceeding this way for n iterations, we end up with a string y^* which is $(1-n\epsilon)$-good. Setting $\epsilon = 1/n^2$, we see that y^* beats at least a fraction of $(1-1/n) \geq 1/2$ columns in X. We have proved Lemma 3.4, modulo Lemma 3.5.

Prefix Ratio Goodness Testing (PRGT): The instances of this promise language have four components: (i) a $2^n \times 2^m$ Boolean matrix M; (ii) a set of columns $X \subseteq \{0,1\}^m$; (iii) a prefix β of length at most n. (iv) parameters $\rho > 0$ and $\epsilon > 0$. The matrix M and the set X are represented succinctly in the form of circuits.
Positive Instances: There exists a row y extending β such that $\mu(y, X) \geq \rho$.
Negative Instances: For all rows y extending β, it is the case that $\mu(y, X) \leq \rho - \epsilon$.

Lemma 3.5. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *that solves the* PRGT *problem. Namely, for positive instances, the output is "yes"; for negative instances, the output is "no"; for other instances, the output can be arbitrary. The running time of the algorithm has a polynomial dependence on* $1/\epsilon$.

3.2. Prefix Ratio Goodness Testing: Proof of Lemma 3.5

In this section, we prove Lemma 3.5. One of the hurdles in trying to construct the desired algorithm is that the gap between the two cases we need to distinguish is small. So, as a first step, we amplify the gap using standard techniques.

The amplification process involves a parameter t, which we will fix suitably. Construct a matrix $\overline{M}$ from M as follows. Each row in $\overline{M}$ corresponds to a row in M and each column $\overline{z}$ in $\overline{M}$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_t \rangle$ of t columns from M. Thus, the matrix $\overline{M}$ is of size $2^n \times 2^{\overline{m}}$, where $\overline{m} = mt$. Consider a row $y \in \{0,1\}^n$ and a column $\overline{z} = \langle z_1, z_2, \ldots, z_t \rangle$, where each z_i is a column in M. Set the entry $\overline{M}[y, \overline{z}] = 1$, if y beats at least $(\rho - \frac{\epsilon}{2})$ fraction of the z_i's (with respect to M); otherwise, set it to 0. Analogously, denote by $\overline{X}$ the t-wise cartesian product of X with itself, i.e., $\overline{X} = \{\langle z_1, z_2, \ldots, z_t \rangle \ : \ z_i \in X\}$. We fix $t = 16m/\epsilon^2$. An application of Chernoff bounds yields the following claim.

Lemma 3.6. *For any* $y \in \{0,1\}^n$, *we have the following.*

- *If* $\mu(y, X) \geq \rho$ *in* M *then* $\mu(y, \overline{X}) \geq 1/2$ *in* $\overline{M}$.
- *If* $\mu(y, X) \leq \rho - \epsilon$ *in* M *then* $\mu(y, \overline{X}) \leq 1/\overline{m}^4$ *in* $\overline{M}$.

Given the above amplification, the problem considered in Lemma 3.5 reduces to the problem addressed in Lemma 3.7. Formally, the algorithm claimed in Lemma 3.5 works as follows. Given a circuit C succinctly encoding the matrix M, a circuit C_X succintly encoding a set of columns X, prefix β and parameters ρ and ϵ, we consider the matrix $\overline{M}$ and the set $\overline{X}$, as described above. Notice that we can construct in polynomial time a circuit $\overline{C}$ succinctly encoding $\overline{M}$ such that $|\overline{C}|$ is polynomial in $|C|$ and $1/\epsilon$. Similarly, we can construct a polynomial size circuit $\overline{C}_X$ succinctly encoding the set $\overline{X}$. Then, we invoke the algorithm given in Lemma 3.7 with $\overline{C}$, $\overline{C}_X$ and β as inputs. We output "yes", if the algorithm outputs "yes" and output "no", otherwise. This completes the proof of Lemma 3.5, modulo Lemma 3.7.

Lemma 3.7. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes three inputs: (i) a* $2^n \times 2^m$ *Boolean matrix* M; *(ii) a set of columns* $X \subseteq \{0,1\}^m$. *(iii) a prefix* β *of length at most* n. *The matrix* M *and the set* X *are presented succinctly in the form of circuits. The algorithm has the following property:*

- *Case (a) : If there exists a row* y *extending* β *such that* $\mu(y, X) \geq 1/2$, *then it outputs "yes".*
- *Case (b) : If all rows* y *extending* β *are such that* $\mu(y, X) \leq 1/m^4$, *then it outputs "no".*

If neither of the above conditions is true, then the output of the algorithm is arbitrary.

There are two main stages in the algorithm. In the first stage we get an estimate on the size of X. And in the second stage, we use the above estimate to distinguish between the cases (a) and (b) in the lemma. Both the stages make queries to a prAM language given as oracle. A lemma, due to Sipser [20], is useful in establishing that the concerned language indeed lies in the class prAM. The following notation is needed for describing the lemma.

Let $\mathcal{H}$ be a family of functions mapping $\{0,1\}^m$ to $\{0,1\}^k$. Recall that $\mathcal{H}$ is said to be 2-universal, if for any $z, z' \in \{0,1\}^m$, with $z \neq z'$, and $x, x' \in \{0,1\}^k$, $\Pr_{h\in\mathcal{H}}[h(z) = x \text{ and } h(z') = x'] = 1/2^{2k}$. It is well known that such a family can easily be constructed. For instance, the set of all $m \times k$ Boolean matrices yield such a family; a matrix B represents the function h given by $h(z) = zB$ (modulo 2).

For a function $h \in \mathcal{H}$ and a string $z \in \{0,1\}^m$, we say that z has a *collision* under h, if there exists a $z' \in \{0,1\}^m$ such that $z \neq z'$ and $h(z) = h(z')$. For a set of hash functions $H \subseteq \mathcal{H}$, we say that z has a *collision* under H, if for all $h \in H$, z has a collision under h. A set $S \subseteq \{0,1\}^m$ is said to have a collision under H, if there exists a $z \in S$ such that z has a collision under H.

Lemma 3.8 ([20])**.** *Let $S \subseteq \{0,1\}^m$ and $k \leq m$. Let $\mathcal{H}$ be a 2-universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^k$. Uniformly and independently pick a set of hash functions $h_1, h_2, \ldots, h_k$ from $\mathcal{H}$ and let $H = \{h_1, h_2, \ldots, h_k\}$. Then,*

- *If $|S| > k2^k$, then $\Pr_H[S \text{ has a collision under } H] = 1$.*
- *If $|S| \leq 2^{k-1}$, then $\Pr_H[S \text{ has a collision under } H] \leq 1/2$.*

We define a promise language called *set largeness testing* (SLT) and then use Lemma 3.8 to show that it lies in the class prAM.

Set Largeness Testing (SLT): The instances in this language consist of a set $X \subseteq \{0,1\}^m$, presented succinctly in the form of a circuit, and an integer $k \leq m$.
Positive instances: $|X| > k2^k$.
Negative instances: $|X| \leq 2^{k-1}$.

Lemma 3.9. *The promise language* SLT *belongs to the class* prAM.

Proof. Let $\mathcal{H}$ be a 2-universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^k$. The proof is based on the observation that for a given set $H \subseteq \mathcal{H}$, testing whether X has a collision under H is an NP predicate.

The AM protocol proceeds as follows. Arthur picks a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ uniformly and independently at random from $\mathcal{H}$. Merlin must exhibit an element $z \in X$ and prove that z has a collision under H. Arthur accepts, if Merlin proves that such a collision exists; otherwise, Arthur rejects. □

The following lemma provides an algorithm for estimating the size of a set, given SLT as oracle.

Lemma 3.10. *There exists an $\mathrm{FP}^{\mathrm{prAM}}$ that takes a set $X \subseteq \{0,1\}^m$, presented succinctly in the form of a circuit, and outputs an estimate U such that $\frac{|U|}{4m} \leq |X| \leq |U|$.*

Proof. The algorithm takes the promise language SLT as the oracle. We iteratively consider every integer k in the range 1 through m and ask the query (X, k) to the oracle. Let k_e be the first time, we get a "no" answer from the oracle. Compute $|U| = m2^{k_e}$. We shall argue that U satisfies the stated bounds.

Let k_0 be the largest integer such that $|X| > k_0 2^{k_0}$ and let k_1 be the smallest integer such that $|X| \leq 2^{k_1-1}$. Notice that $k_0 + 1 \leq k_e \leq k_1$. By the property of k_0, k_e satisfies $|X| \leq k_e 2^{k_e} \leq m2^{k_e}$. By the property of k_1, we have that $2^{k_1-2} < |X| \leq 2^{k_1-1}$. It follows that $2^{k_e} \leq 2^{k_1} < 4|X|$. The claimed bounds on $|U|$ follow from the above inequalities. □

Returning to Lemma 3.7, the first stage of the algorithm (finding an estimate on $|X|$) can now be performed using Lemma 3.10. We turn to the second stage that involves distinguishing between the two cases in Lemma 3.7. For this, we will make use of the following promise language as an oracle.

Prefix Cardinality Goodness Testing (PCGT): The instances of this language consist of four components: (i) a $2^n \times 2^m$ Boolean matrix M; (ii) a set $X \subseteq \{0,1\}^m$; (iii) a prefix β of length at most n; (iv) a number k. The matrix M and the set X are presented succinctly in the form of circuits.

Positive instances: There exists a row y extending β such that y beats at least $k2^k$ columns in X.

Negative instances: For all rows y extending β, y beats at most 2^{k-1} columns in X.

Lemma 3.11. *The promise language* PCGT *belongs to the class* prAM.

Proof. The proof is similar to that of Lemma 3.9 and makes use of Lemma 3.8. We present an MAM protocol. It is well known that such a protocol can be converted to an AM protocol [3].

Merlin claims that a given instance is of the positive type. To prove this, he provides a row y extending β. Let $Z \subseteq X$ be the set of columns from X that are beaten by y. Arthur needs to distinguish between the cases of $|Z| > k2^k$ and $|Z| \leq 2^{k-1}$. This situation is the same as that of Lemma 3.9. By repeating the argument from there, we get an MAM protocol. □

Proof of Lemma 3.7: Our algorithm will make use of both SLT and PCGT as oracles. Let us rephrase the two cases that we wish to distinguish:

- Case (a): There exists a row y extending β such that y beats at least $|X|/2$ columns from X.
- Case (b) : For any row y extending β, y beats at most $|X|/m^4$ columns from X.

We first run the algorithm claimed in Lemma 3.10 to get an estimate U such that $|U|/4m \leq |X| \leq |U|$. Our next goal is to reduce the task of distinguishing the above two cases to a PCGT query. Consider any row y. Let Z be the number of columns from X that y beats. We wish to choose a number k satisfying two conditions: (i) if $Z \geq |X|/2$ then $Z > k2^k$; (ii) if $Z \leq |X|/m^4$ then $Z \leq 2^{k-1}$. A simple calculation reveals that it suffices for k to satisfy the following inequalities in terms of U:

$$\frac{2U}{m^4} \leq 2^k \leq \frac{U}{8m^2}.$$

Clearly, we can choose $k = \lfloor \log \frac{U}{8m^2} \rfloor$. Then, we call the PCGT oracle with the parameters M, X, β and k. We output "yes", if the oracle says "yes"; and output "no", if the oracle says "no". □

4. Applications of the Main Result

In this section, we apply Theorem 3.2 in two different settings and derive some corollaries. The first deals with upperbounds on the power of S_2^p. The second is about the consequences of NP having polynomial size circuits.

4.1. Upperbounds for S_2^p

Theorem 4.1. $S_2^p \subseteq P^{prAM}$.

Proof. The claim follows directly from Theorem 3.2. Let L be a language in S_2^p. Let x be the input string. Consider the S_2-type matrix M corresponding to x. As discussed in Section 2, we can obtain a circuit C succinctly encoding the matrix M in time polynomial in $|x|$. Invoking the algorithm given in Theorem 3.2 on C, we get either a row-side `CIC` or a column-side `CIC`. Notice that in the former case $x \in L$ and in the latter case $x \notin L$. □

Having proven the above theorem, it is natural to ask how large the class P^{prAM} is. By definition, AM is contained in BPP^{NP} and so, P^{AM} is also contained in the same class. We observe the above claim extends to the case where the oracle is a prAM oracle, instead of an AM oracle.

Theorem 4.2. $P^{prAM} \subseteq BPP^{NP}$.

Cai [5] showed that S_2^p is contained in ZPP^{NP}, whereas our result puts S_2^p in the class P^{prAM}. The containment relationships between ZPP^{NP} and P^{prAM} are unknown. In this context, we observe that an alternative proof of Cai's result can be derived using our techniques. This cannot be achieved by simply combining Theorem 4.1 and 4.2; this would only yield $S_2^p \subseteq BPP^{NP}$. We obtain the alternative proof by directly appealing to Theorem 3.2.

Theorem 4.3 ([5])**.** $S_2^p \subseteq ZPP^{NP}$.

4.2. Consequences of NP having small circuits

A body of prior work has dealt with the implication of the assumption that NP has polynomial size circuits. Our main theorem yields some new results in this context, which are described in this section.

Suppose NP is contained in P/poly. Karp and Lipton [13] showed that, under this assumption, the polynomial time hierarchy (PH) collapses to $\Sigma_2^p \cap \Pi_2^p$, i.e., $PH = \Sigma_2^p \cap \Pi_2^p$. Köbler and Watanabe [14] improved the collapse to ZPP^{NP}. Sengupta (see [5]) observed that the collapse can be brought down to S_2^p. This has been further improved via a collapse to O_2^p, the oblivious version of S_2^p [7]. It has been an interesting open problem to obtain a collapse to the class P^{NP}. Here, we show a collapse to P^{prMA}.

Theorem 4.4. *If* $NP \subseteq P/poly$*, then* $PH = P^{prMA}$.

Proof. By Sengupta's observation [5], the assumption implies that $PH = S_2^p$. Combining this with Theorem 4.1, we get $PH = P^{prAM}$. Arvind et al. [2] showed that if $NP \subseteq P/poly$ then $AM = MA$. We observe that this result carries over to the promise versions, namely the same assumption implies $prAM = prMA$. The claim follows. □

Though the above theorem yields a new consequence, we note that it is not clear whether this is an improvement over the previously best known collapse. It is known that $MA \subseteq ZPP^{NP}$ [11, 1] and $MA \subseteq S_2^p$ [17]. Extending the former claim, we can show that $P^{prMA} \subseteq ZPP^{NP}$. However, we do not know how to accomplish the same for the second claim. Namely, it remains open whether P^{prMA} is contained in S_2^p.

Under the assumption NP has polynomial size circuits, Bshouty et al. [4] studied the problem of learning a correct circuit for SAT and designed a ZPP^{NP} algorithm. Using Theorem 3.2, we derive the following claim which improves the above result, as we can show that $FP^{prMA} \subseteq ZPP^{NP}$.

Theorem 4.5. *If* NP $\subseteq$ P/poly*, then there exists an* $\mathrm{FP}^{\mathrm{prMA}}$ *algorithm that outputs a correct polynomial size circuit for* SAT *at a given input length.*

Acknowledgments: We thank the anonymous referees for their useful comments.

References

[1] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1997.

[2] V. Arvind, J. Köbler, U. Schöning, and R. Schuler. If NP has polynomial-size circuits, then MA=AM. *Theoretical Computer Science*, 137(2):279–282, 1995.

[3] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[4] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.

[5] J. Cai. $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. *Journal of Computer and System Sciences*, 73(1), 2007.

[6] R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.

[7] V. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, 2006.

[8] D. Du and K. Ko. *Computational Complexity*. John Wiley and sons, 2000.

[9] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans. On the complexity of succinct zero-sum games. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, 2005.

[10] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In *RANDOM-APPROX*, 1999.

[11] O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). Technical Report TR97–045, ECCC, 1997.

[12] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43, 1986.

[13] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, 1980.

[14] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.

[15] P. Miltersen and N. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.

[16] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[17] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.

[18] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, 2007.

[19] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4), 2007.

[20] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983.

[21] L. Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983.