# Arthur and Merlin as Oracles *

Venkatesan T. Chakaravarthy Sambuddha Roy

IBM India Research Lab, New Delhi, India
{vechakra, sambuddha}@in.ibm.com

**Abstract**

We study some problems solvable in deterministic polynomial time given oracle access to the (promise version of) the Arthur-Merlin class. Our main results are the following:

- $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$
- $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$

In addition to providing new upperbounds for the classes $\mathrm{S}_2^p$ and $\mathrm{BPP}_{||}^{\mathrm{NP}}$, these results are interesting from a derandomization perspective. In conjunction with the hitting set generator construction of Miltersan and Vinodchandran [22], we get that $\mathrm{S}_2^p = \mathrm{P}^{\mathrm{NP}}$ and $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$, under the hardness hypothesis associated with derandomizing the class AM. This gives an alternative proof of the same result obtained by Shaltiel and Umans [29].

We also show that if NP has polynomial size circuits then the polynomial time hierarchy (PH) collapses as $\mathrm{PH} = \mathrm{P}^{\mathrm{prMA}}$. Under the same hypothesis, we also derive an $\mathrm{FP}^{\mathrm{prMA}}$ algorithm for learning circuits for SAT; this improves the $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the same problem by Bshouty et al. [5].

Finally, we design a $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for the problem of finding near-optimal strategies for succinctly presented zero-sum games. For the same problem, Fortnow et al. [13] described a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm. One advantage of our $\mathrm{FP}^{\mathrm{prAM}}$ algorithm is that it can be derandomized using the construction of Miltersen and Vinodchandran [22] yielding a $\mathrm{FP}^{\mathrm{NP}}$ algorithm, under a hardness hypothesis used for derandomizing AM.

## 1 Introduction

We study some problems solvable in deterministic polynomial time given oracle access to the (promise version of) the Arthur-Merlin class, namely the class $\mathrm{P}^{\mathrm{prAM}}$ and its variants, such as $\mathrm{P}_{||}^{\mathrm{prAM}}$. Our main results are the following:

- $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$
- $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$

In addition to providing new upperbounds for the classes $\mathrm{S}_2^p$ and $\mathrm{BPP}_{||}^{\mathrm{NP}}$, these results are interesting from a derandomization perspective.

*Parts of this paper appear in the proceedings of STACS'08 [10] and MFCS'08 [9].

**Derandomization Perspective:** Starting with the classical hardness-randomness tradeoff due to Nisan and Wigderson [25], several complexity classes have been derandomized, under various hardness hypotheses. In this framework, we derandomize a probabilistic class $\mathcal{C}$ (such as AM) under a suitable hardness hypothesis. A typical hardness hypothesis assumes the existence of a language $L$ contained in a suitable uniform complexity class (such as $\mathrm{NE} \cap \mathrm{coNE}$) that cannot be computed by circuits of a specific type of "small" size (such as SV-nondeterministic circuits of size $2^{\epsilon n}$, for some $\epsilon > 0$). From such a language $L$, one then constructs an "efficient" pseudorandom generator, which is used to derandomize the class $\mathcal{C}$ under consideration. The circuit type in the hypothesis is determined based on the complexity class $\mathcal{C}$ that we wish to derandomize.

Working under the above framework, Klivans and van Melkebeek [19] derandomized the class $\mathrm{BPP}_{||}^{\mathrm{NP}}$ and showed that $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$, under a hardness hypothesis naturally associated with the class $\mathrm{BPP}_{||}^{\mathrm{NP}}$, which we shall refer to as the $\mathrm{BPP}_{||}^{\mathrm{NP}}$-hypothesis [1]. They also derandomized the class $\mathrm{BPP}^{\mathrm{NP}}$ and showed that $\mathrm{BPP}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}$, under a hardness hypothesis naturally associated with the class $\mathrm{BPP}^{\mathrm{NP}}$, which we shall refer to as the $\mathrm{BPP}^{\mathrm{NP}}$-hypothesis [2]. They obtained these results by building on the work of Impagliazzo and Wigderson [17]

Cai [6] showed that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ and it is known that $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{S}_2^p$ [28]. An important open problem regarding $\mathrm{S}_2^p$ is whether $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$. Notice that by combining Cai's result [6] with that of Klivans and Melkebeek [19], we get that $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$, under the $\mathrm{BPP}^{\mathrm{NP}}$-hypothesis.

To summarize the discussion so far, we have that $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$, under $\mathrm{BPP}_{||}^{\mathrm{NP}}$-hypothesis and $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$, under $\mathrm{BPP}^{\mathrm{NP}}$-hypothesis. In this context, Shaltiel and Umans [29] obtained an interesting improvement by deriving both the conclusions above, a under weaker hypothesis. They showed that $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$ and $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$, under a hypothesis which we shall refer to as the AM-hypothesis [3]. The AM-hypothesis is naturally associated with derandomizing the class AM. This hypothesis was introduced by Miltersen and Vinodchandran [22], who showed that $\mathrm{AM} = \mathrm{NP}$, under this hypothesis. Shaltiel and Umans [29], in fact, show that the $\mathrm{BPP}_{||}^{\mathrm{NP}}$-hypothesis and the AM-hypothesis are equivalent. We primarily focus on the following corollaries of the above result: $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$ and $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$, under the AM-hypothesis. This is surprising, since they derandomize $\mathrm{BPP}_{||}^{\mathrm{NP}}$ and $\mathrm{S}_2^p$, under a weaker hypothesis associated with the smaller class $\mathrm{AM} \subseteq \mathrm{BPP}_{||}^{\mathrm{NP}}$.

Our main result yields an alternative (and, perhaps more direct) proof of these derandomization results. The alternative proof is obtained by combining our main results with the hitting set generator construction of Miltersen and Vinodchandran [22]. As a direct consequence of the above construction, we get that $\mathrm{P}^{\mathrm{prAM}} \subseteq \mathrm{P}^{\mathrm{NP}}$ and $\mathrm{P}_{||}^{\mathrm{prAM}} \subseteq \mathrm{P}_{||}^{\mathrm{NP}}$, under the AM-hypothesis. Combined with our main results, we get that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{NP}}$ and $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{NP}}$, under AM-hypothesis. In this alternative proof, it is interesting to note that the construction of Miltersen and Vinodchandran is used almost as a black-box.

**Corollaries of the Main Results:** Our result that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$ yields the following corollaries. First, we show that $\mathrm{BPP}_{||}^{\mathrm{prAM}} = \mathrm{P}_{||}^{\mathrm{prAM}}$; this is an unconditional derandomization of the class $\mathrm{BPP}_{||}^{\mathrm{prAM}}$. Second, we derive that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\Sigma_2^p}$, or equivalently $\mathrm{BPP}^{\mathrm{NP}[\log]} \subseteq \mathrm{P}^{\Sigma_2^p[\log]}$. This

[1] $\mathrm{BPP}_{||}^{\mathrm{NP}}$-hypothesis: there exists a language $L$ computable in $\mathrm{NE} \cap \mathrm{coNE}$ and an $\epsilon > 0$ such that for sufficiently large $n$, non-adaptive SAT-oracle circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$.

[2] $\mathrm{BPP}^{\mathrm{NP}}$-hypothesis: there exists a language $L$ computable in $\mathrm{NE} \cap \mathrm{coNE}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SAT-oracle circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$.

[3] AM-hypothesis: there exists a language $L$ computable in $\mathrm{NE} \cap \mathrm{coNE}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$.

may be seen as a baby-step towards resolving the much larger open problem of whether $\mathrm{BPP}^{\mathrm{NP}}$ is contained in $\mathrm{P}^{\Sigma_2^p}$.

Our result that $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$ (or an application of its proof) yields the following corollaries. The first corollary is regarding the classical Karp-Lipton theorem that deals with the consequences of the assumption that NP has polynomial size circuits. Under this assumption, Karp and Lipton [18] showed that the polynomial time hierarchy (PH) collapses to $\Sigma_2^p$. Subsequently, their result has been strengthened: Köbler and Watanabe [20] derived the collapse $\mathrm{PH} = \mathrm{ZPP}^{\mathrm{NP}}$; Sengupta observed that $\mathrm{PH} = \mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ (see [6]); recently, the collapse has been further improved as $\mathrm{PH} = \mathrm{O}_2^p \subseteq \mathrm{S}_2^p$ [8]. It has been a challenging open problem to get the collapse down to $\mathrm{P}^{\mathrm{NP}}$. We derive a weaker result: if NP has polynomial size circuits, then $\mathrm{PH} = \mathrm{P}^{\mathrm{prMA}}$.

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. [5] designed a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm that finds a correct circuit for SAT at a given length. For the same problem, we present a $\mathrm{FP}^{\mathrm{prMA}}$ algorithm for the same task. We can show that $\mathrm{FP}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ and hence, our result is an improvement over the result of Bshouty et al.

**Succinct Zero-sum Games:** By extending the ideas from the proof of the result $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$, we derive an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding near optimal strategies for succinct two-player zero-sum games.

Zero-sum games have been well explored, due to their diverse applications. The problem of finding the value, as well as optimal and near-optimal strategies of a given game have been well-studied. In particular, it is known that the value and optimal strategies can be computed in polynomial time (see [26]). We refer to [13] for a brief account of these results and the applications of zero-sum games in computational complexity and learning theory.

In this paper, we deal with computing near-optimal strategies when the payoff matrix $M$ is presented *succinctly* in the form of a circuit $C$. It is known that computing the exact value of a succinctly presented zero-sum game is EXP-complete (see [12]) and that approximating the value within multiplicative factors is $\Pi_2^p$-hard [13]. Lipton and Young [21] showed that near-optimal strategies, within additive errors, of a succinctly presented zero-sum game can be computed in $\Sigma_2^p$. Fortnow et. al [13] presented a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the same problem; their algorithm also finds an estimate of the value of the game within additive errors. The above problem generalizes the problem of learning circuits for SAT (assuming such circuits exist) and the problems in the symmetric alternation class $\mathrm{S}_2^p$. We refer the reader to the paper by Fortnow et. al [13] for a detailed account on these aspects.

Here, our main result is an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for the above problem, namely the problem of finding near-optimal strategies, within additive errors, of a succinctly presented zero-sum game. The algorithm utilizes ideas from a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm due to Cai [6] for finding "strong irrefutable certificates".

As discussed earlier, the Miltersen-Vinodchandran construction [22] can be directly used to derandomize the $\mathrm{FP}^{\mathrm{prAM}}$ algorithm. Thus, under the AM-hypothesis discussed earlier, we get an $\mathrm{FP}^{\mathrm{NP}}$ algorithm for finding near optimal strategies. It is not clear whether such a derandomization can be achieved for the $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm by Fortnow et. al [13].

**Alternative Proofs:** Using our main results, we derive alternative proofs for the following results: (i) Cai's result [6] that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$; (ii) The result by Bshouty et al. [5] that if NP has polynomial size circuits, then circuits for SAT can be found in $\mathrm{ZPP}^{\mathrm{NP}}$; (iii) The result by Fortnow et al. that near-optimal strategies of succinctly represented zero-sum games can be computed in

$\mathrm{ZPP}^{\mathrm{NP}}$; (iv) A lemma by Fortnow et. al [14] that provides a mechanism for proving that SAT does not have small size circuits, if it is the case (see Appendix).

## 2 Preliminaries

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

**Complexity classes.** We use standard definitions for complexity classes such as P, NP, P/poly, MA, AM, $\mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{BPP}^{\mathrm{NP}}$ [11, 27]. Below, we present definitions for promise and function classes central to our paper.

**Promise languages.** A promise language $\Pi$ is a pair $(\Pi_1, \Pi_2)$, where $\Pi_1, \Pi_2 \subseteq \Sigma^*$, such that $\Pi_1 \cap \Pi_2 = \emptyset$. The elements of $\Pi_1$ are called the *positive instances* and those of $\Pi_2$ are called the *negative instances*. When $\Pi_1 \cup \Pi_2 = \Sigma^*$, we get the usual notion of languages.

**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 $\Pi_1$ nor in $\Pi_2$, 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 $\Pi$, the notation $\mathrm{FP}^{\Pi}$ refers to the class of functions that are computable by a polynomial time machine, given oracle access to $\Pi$. For a promise class $\mathcal{C}$, we denote by $\mathrm{FP}^{\mathcal{C}}$, the union of $\mathrm{FP}^{\Pi}$, 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 $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$

In this section, we prove that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$. We start with the definition of $\mathrm{BPP}_{||}^{\mathrm{NP}}$.

**Definition 3.1** *A language $L$ is said to be in the class $\mathrm{BPP}_{||}^{\mathrm{NP}}$, if there exists a $\mathrm{P}_{||}^{\mathrm{NP}}$ machine $M$ and a polynomial $p(\cdot)$ such that for any input $x$,*

$$\begin{aligned} x \in L &\implies \Pr_{y \in \{0,1\}^m}[M(x,y)=1] \geq \frac{3}{4}, \text{ and} \\ x \notin L &\implies \Pr_{y \in \{0,1\}^m}[M(x,y)=1] \leq \frac{1}{4}, \end{aligned}$$

*where $m = p(|x|)$.*

The proof of the claim that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$ goes via approximate counting for non-deterministic circuits, a problem solvable in $\mathrm{P}_{||}^{\mathrm{prAM}}$.

A *non-deterministic* circuit $C$ is a Boolean circuit that takes an $m$-bit string as input and an $s$-bit string $z$ as auxiliary input and outputs 1 or 0, i.e., $C : \{0,1\}^m \times \{0,1\}^s \to \{0,1\}$. For a string $y \in \{0,1\}^m$, $C$ is said to *accept* $y$, if there exists a $z \in \{0,1\}^s$ such that $C(y,z) = 1$; $C$ is said to *reject* $y$, otherwise. Let $\mathrm{Count}(C)$ denote the number of strings from $\{0,1\}^m$ accepted by $C$.

The result below follows from the work of Sipser [30] and Stockmeyer [31] and it deals with the problem of approximately counting the number of strings accepted by a given non-deterministic circuit.

**Theorem 3.2** *[30][31] There exists an $\mathrm{FP}_{||}^{\mathrm{prAM}}$ algorithm that takes as input a non-deterministic circuit $C$, and a parameter $\delta > 0$ and outputs a number $U$ such that*

$$(1-\delta)U \leq \mathrm{Count}(C) \leq (1+\delta)U.$$

*The running time is polynomial in $|C|$ and $1/\delta$.*

The first step of the proof involves reducing the error probability of a given $\mathrm{BPP}_{||}^{\mathrm{NP}}$ machine. This is achieved via the standard method of repeated trials and taking majority.

**Proposition 3.3** *Let $L$ be a language in $\mathrm{BPP}_{||}^{\mathrm{NP}}$. Then there exists a $\mathrm{P}_{||}^{\mathrm{NP}}$ machine $M$ and a polynomial $p(\cdot)$ such that for any input $x$,*

$$\begin{aligned} x \in L &\implies \Pr_{y \in \{0,1\}^m}[M(x,y)=1] \geq 1 - \frac{1}{8K}, \text{ and} \\ x \notin L &\implies \Pr_{y \in \{0,1\}^m}[M(x,y)=1] \leq \frac{1}{8K}, \end{aligned}$$

*where $m = p(|x|)$ and $K \leq p(|x|)$ is the maximum number of queries asked by $M$ on input $x$ for any string $y \in \{0,1\}^m$.*

**Theorem 3.4** $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$

*Proof:* Let $L$ be a language in $\mathrm{BPP}_{||}^{\mathrm{NP}}$ and let $M$ be a $\mathrm{BPP}_{||}^{\mathrm{NP}}$ machine for deciding $L$ given by Proposition 3.3. Fix an input string $x$ and let $m$ be the number of random bits used by $M$. Without loss of generality, assume that $M$ uses SAT as its oracle and that the number of queries is exactly $K$ on all random strings $y \in \{0,1\}^m$, with $K \geq 1$.

Partition the set $\{0,1\}^m$ into the set of *good* strings $G$ and the set of *bad* strings $B$, where $G = \{y \ : \ M(x,y) = \chi_L(x)\}$ and $B = \{0,1\}^m - G$, where $\chi_L(\cdot)$ is the characteristic function [4]. For a set $X \subseteq \{0,1\}^m$, let $\mu(X) = |X|/2^m$ denote its measure. Thus, $\mu(G) \geq 1 - 1/(8K)$ and $\mu(B) \leq 1/(8K)$.

Consider a string $y \in \{0,1\}^m$. Let $\Phi(y) = \langle \varphi_1^y, \varphi_2^y, \ldots, \varphi_K^y \rangle$, be the $K$ SAT queries asked by $M$ on the string $y$. Let $\underline{\mathbf{a}}^y = \langle a_1^y, a_2^y, \ldots, a_K^y \rangle$ be the correct answers to these queries, namely the bit $a_j^y = 1$, if $\varphi_j^y \in$ SAT and $a_j^y = 0$, otherwise. We shall consider simulating the machine $M$ with arbitrary answer strings. For a bit string $\underline{\mathbf{b}} = \langle b_1 b_2 \ldots b_K \rangle$, let $M(x, y, \underline{\mathbf{b}})$ denote the outcome of the machine, if $\underline{\mathbf{b}}$ is provided as the answer to its $K$ queries. Thus, $M(x,y) = M(x,y,\underline{\mathbf{a}}^y)$. When an arbitrary bit-string $\underline{\mathbf{b}}$ is provided as the answer, the outcome $M(x,y,\underline{\mathbf{b}})$ can be different from $M(x,y)$.

Let $N(y)$ denote the number of satisfiable formulas in $\Phi(y)$, i.e., $N(y) = |\{1 \leq i \leq K : \ \varphi_i^y \in \text{SAT}\}|$. Partition the set $G$ into $K+1$ parts based on the value $N(\cdot)$: for $0 \leq r \leq K$, let $S_r = \{y \ : \ y \in G \text{ and } N(y) = r\}$.

For each $0 \leq r \leq K$, define two sets $C_r^1$ and $C_r^0$ as below. The set $C_r^1$ consists of all strings $y \in \{0,1\}^m$ satisfying the following property: there exists an answer string $\underline{\mathbf{b}} = \langle b_1 b_2 \ldots b_K \rangle$ such that (i) if $b_j = 1$ then $\varphi_j^y \in$ SAT; (ii) $\underline{\mathbf{b}}$ has at least $r$ ones in it; and (iii) $M(x,y,\underline{\mathbf{b}}) = 1$. The set $C_r^0$ is defined similarly. The set $C_r^0$ consists of all strings $y \in \{0,1\}^m$ satisfying the following property: there exists an answer string $\underline{\mathbf{b}} = \langle b_1 b_2 \ldots b_K \rangle$ such that (i) if $b_j = 1$ then $\varphi_j^y \in$ SAT; (ii) $\underline{\mathbf{b}}$ has at least $r$ ones in it; and (iii) $M(x,y,\underline{\mathbf{b}}) = 0$. Notice that the sets $C_r^1$ and $C_r^0$ need not be disjoint and that there may be strings $y \in \{0,1\}^m$ that belong to neither $C_r^1$ nor $C_r^0$.

**Claim 1:**
(i) If $x \in L$, then there exists an $0 \leq \ell \leq K$ such that $\mu(C_\ell^1) - \mu(C_\ell^0) \geq \frac{1}{4K}$.
(ii) If $x \notin L$, then for all $0 \leq r \leq K$, $\mu(C_r^1) - \mu(C_r^0) \leq \frac{1}{8K}$.

*Proof of claim:* We first make an observation regarding the concerned sets $C_r^1$ and $C_r^0$. Fix any $0 \leq r \leq K$. Notice that for any $j < r$, $S_j \cap C_r^1 = \emptyset$ and $S_j \cap C_r^0 = \emptyset$. This is because, for any $y \in S_j$, with $j < r$, $\Phi(y)$ has only $j$ satisfying formulas and so, it does not meet the requirement for either $C_r^1$ or $C_r^0$.

Suppose $x \in L$. Consider any $0 \leq r \leq K$. We first derive a lowerbound on $|C_r^1|$. Consider any string $y \in S_j$, with $r \leq j \leq K$. Notice that $\underline{\mathbf{a}}^y$ has $j \geq r$ satisfying formulas and $M(x,y,\underline{\mathbf{a}}^y) = 1$. So, $y \in C_r^1$. Thus, for $r \leq j \leq K$, $S_j \subseteq C_r^1$. It follows that $|C_r^1| \geq \sum_{j=r}^K |S_j|$. We next derive an upperbound on $C_r^0$. We observed that $S_j \cap C_r^0 = \emptyset$, for all $j < r$. Let us now consider the set $S_r$. Pick any string $y \in S_r$. Notice that $\Phi(y)$ has exactly $r$ satisfiable formulas. So, the only answer string satisfying the first two requirements of $C_r^0$ is $\underline{\mathbf{a}}^y$. Since $M(x,y,\underline{\mathbf{a}}^y) = 1$, we have that $y \notin C_r^0$. Thus, $S_r \cap C_r^0 = \emptyset$. Hence, for $j \leq r$, $S_j \cap C_r^0 = \emptyset$. It follows that $|C_r^0| \leq |B| + \sum_{j=r+1}^K |S_j|$. As a consequence, we get that $|C_r^1| - |C_r^0| \geq |S_r| - |B|$. Since $\mu(G) \geq 1 - 1/(8K)$, by an averaging argument, there exists an $0 \leq \ell \leq K$ such that

$$\mu(S_\ell) \geq \frac{\left(1 - \frac{1}{8K}\right)}{K+1} \geq \frac{3}{8K}.$$

Such an $\ell$ satisfies $\mu(C_\ell^1) - \mu(C_\ell^0) \geq 1/(4K)$. We have proved the first part of the claim.

Suppose $x \notin L$. The argument is similar to the first part. Fix any $0 \leq r \leq K$. Observe that for $j \leq r$, $S_j \cap C_r^1 = \emptyset$. Hence, $|C_r^1| \leq |B| + \sum_{j=r+1}^K |S_j|$. On the other hand, for any $j \geq r$, $S_j \subseteq C_r^0$.

[4] $\chi_L(x) = 1$, if $x \in L$ and $\chi_L(x) = 0$, if $x \notin L$

So, $|C_r^0| \geq \sum_{j=r}^{K} |S_j|$. It follows that $|C_r^1| - |C_r^0| \leq |B|$. Since $\mu(B) \leq 1/(8K)$, we get the second part of the claim. This completes the proof of Claim 1.

Notice that the membership testing for the sets $C_j^1$ and $C_j^0$ can be performed in non-deterministic polynomial time. Thus, for $0 \leq j \leq K$, we can construct a non-deterministic circuit accepting $C_j^1$ (similarly, $C_j^0$) such that the size of the circuit is polynomial in $|x|$. Setting $\delta = 1/(80K)$, we invoke the algorithm given by Theorem 3.2 to get estimates $e_j^1$ and $e_j^0$ such that $(1-\delta)\mu(C_j^1) \leq e_j^1 \leq (1+\delta)\mu(C_j^1)$ and $(1-\delta)\mu(C_j^0) \leq e_j^0 \leq (1+\delta)\mu(C_j^0)$, for all $0 \leq j \leq K$. From Claim 1, we get: (i) if $x \in L$, then there exists an $0 \leq \ell \leq K$ such that $e_\ell^1 - e_\ell^0 \geq 9/(40K)$; (ii) if $x \notin L$ then for all $0 \leq j \leq K$, $e_j^1 - e_j^0 \leq 6/(40K)$. So, we output "$x \in L$", if there exists an $\ell$ such that $e_\ell^1 - e_\ell^0 \geq 9/(40K)$. If no such $\ell$ exists, then we output "$x \notin L$". □

As a corollary, we get that $\mathrm{BPP}_{||}^{\mathrm{prAM}} = \mathrm{P}_{||}^{\mathrm{prAM}}$. This is an unconditional derandomization of the class $\mathrm{BPP}_{||}^{\mathrm{prAM}}$.

**Corollary 3.5** $\mathrm{BPP}_{||}^{\mathrm{prAM}} = \mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{prAM}}$.

*Proof:* It is easy to show that $\mathrm{BPP}_{||}^{\mathrm{prAM}} \subseteq \mathrm{BPP}_{||}^{\mathrm{NP}}$. From Theorem 3.4, we have that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$. It is trivially true that $\mathrm{P}_{||}^{\mathrm{prAM}} \subseteq \mathrm{BPP}_{||}^{\mathrm{prAM}}$. □

As a second corollary, we get the following result. It may be seen as a baby-step towards resolving the larger open problem of whether $\mathrm{BPP}^{\mathrm{NP}}$ is contained in $\mathrm{P}^{\Sigma_2^p}$.

**Corollary 3.6** $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\Sigma_2^p}$ *or equivalently,* $\mathrm{BPP}^{\mathrm{NP[log]}} \subseteq \mathrm{P}^{\Sigma_2^p[\log]}$

*Proof:* By extending the proof of $\mathrm{AM} \subseteq \Pi_2^p$ [4], it is easy to show that $\mathrm{P}_{||}^{\mathrm{prAM}} \subseteq \mathrm{P}_{||}^{\Sigma_2^p}$. It follows that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\Sigma_2^p}$. It is known that $\mathrm{P}_{||}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP[log]}}$ [27]. Using the same idea, it can be shown that that $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{BPP}^{\mathrm{NP[log]}}$ and that $\mathrm{P}_{||}^{\Sigma_2^p} = \mathrm{P}^{\Sigma_2^p[\log]}$. □

# 4 $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$ and Other Results

In this section, we prove that $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$ and derive some corollaries. We start with an informal description of the class $\mathrm{S}_2^p$ and introduce some necessary definitions.

The class $\mathrm{S}_2^p$ was introduced by Russell and Sundaram [28] and independently, by Canetti [7]. A language $L$ in the class $\mathrm{S}_2^p$ 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 otherwise, we say that $z$ *beats* $y$. 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. A formal definition of $S_2^p$ follows next.

**Definition 4.1** *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.

$L \in S_2^p$. The behavior of the verifier $V$ on an input string $x$ can be conveniently modeled as a matrix. 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 as 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 $z \in \{0,1\}^m$, we set $M[y,z] = V(x,y,z)$. Notice that the matrix $M$ has either a row full of 1's (this happens when $x \in L$) or a column full of 0's (this happens when $x \notin L$). We call such matrices as $S_2$-type matrices. $M$ is said to be the *matrix corresponding* to the input $x$. The above discussion is summarized in the following definition.

**Definition 4.2** *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 $y \in \{0,1\}^n$. 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. If $M$ has a row-side IC, it is called a row-side $S_2$-type matrix; similarly, if $M$ has a column-side IC, it is called a column-side $S_2$-type matrix;*

For an input $x$, the matrix $M$ corresponding to $x$ is an $S_2$-type matrix. Though the matrix $M$ is exponentially large in the size of the input $|x|$, it can be *succinctly represented* in the form of a Boolean circuit $C$ having size polynomial in $|x|$.

**Definition 4.3** *A Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is said to* succinctly represent *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 $C(y,z) = M[y,z]$.*

We shall also use the notion of circuits succinctly representing subsets.

**Definition 4.4** *A Boolean circuit $C : \{0,1\}^m \to \{0,1\}$ is said to* succinctly represent *a set $X \subseteq \{0,1\}^m$, if for all $x \in \{0,1\}^m$, $x \in X \iff C(x) = 1$.*

Let $x$ be an input string and $M$ be the matrix corresponding to it. Using standard techniques, we can obtain a Boolean circuit $C$ that succinctly represents the matrix $M$. The above task can be performed in time polynomial in $|x|$. The size of the circuit will be polynomial in $|x|$.

Suppose we wish to prove Cai's result that $S_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ [6] or our main result that $S_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$. Consider an input string $x$. As discussed above, we can find in polynomial time a circuit succinctly

representing the matrix $M$ corresponding to $x$. A natural idea to determine whether $x \in L$ or $x \notin L$ is to actually find a row-side IC or a column-side IC of $M$; this would tell us whether or not $x$ belongs to $L$. Unfortunately, we show that if an irrefutable certificate can be found in $\mathrm{ZPP}^{\mathrm{NP}}$ or $\mathrm{FP}^{\mathrm{prAM}}$, then the polynomial time hierarchy (PH) collapses. We formalize the problem and prove the above claim.

**Problem** FINDIC**:** Given a circuit succinctly representing a $\mathrm{S}_2$-type matrix of size $2^n \times 2^m$, output either a row-side IC or a column-side IC.

**Theorem 4.5** *If there exists a* $\mathrm{ZPP}^{\mathrm{NP}}$ *algorithm or a* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm for the* FINDIC *problem then* $\mathrm{PH} = \mathrm{BPP}^{\mathrm{NP}}$.

*Proof:* It is trivially true that $\mathrm{ZPP}^{\mathrm{NP}}$ algorithms can be simulated in $\mathrm{BPP}^{\mathrm{NP}}$. Furthermore, it is easy to show that $\mathrm{FP}^{\mathrm{prAM}}$ algorithms can be simulated in $\mathrm{BPP}^{\mathrm{NP}}$. Thus, the hypothesis of the theorem implies a $\mathrm{BPP}^{\mathrm{NP}}$ algorithm $\mathcal{A}$ for the FINDIC problem. We show that the existence of the algorithm $\mathcal{A}$ implies that $\Sigma_2^p \subseteq \mathrm{RP}^{\mathrm{NP}}$.

Let $L$ be a language in $\Sigma_2^p$. There exists a polynomial time computable predicate $D(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that for any string $x$,

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

where $n = p(|x|)$ and $m = q(|x|)$.

Consider an input string $x$. Represent the computation of the predicate $D$ in the form of a Boolean matrix $M$ of size $2^n \times 2^m$, by setting $M[y,z] = D(x,y,z)$, for all $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$. Using standard techniques, we can construct a circuit $C$ succinctly representing the matrix $M$ in time polynomial in $|x|$.

We simulate the algorithm $\mathcal{A}$ on the matrix $M$. Consider the two cases of $x \in L$ and $x \notin L$. If $x \in L$, then $M$ is an $\mathrm{S}_2$-type matrix having a row-side IC and so, the algorithm must output a row-side IC with high probability. If $x \notin L$, $M$ need not be an $\mathrm{S}_2$-type matrix and so, the output of the algorithm can be arbitrary. In either case, let $s$ denote the output of $\mathcal{A}$ on input $M$. Using the NP oracle, check if $s$ is a row-side IC. If so, accept $x$; otherwise, reject $x$.

Let us call the above algorithm $\mathcal{B}$. Notice that if $x \in L$, then $\mathcal{B}$ accepts $x$ with high probability. On the other hand, if $x \notin L$, then $\mathcal{B}$ rejects $x$ with probability 1; because, $M$ does not have a row-side IC. Thus, we have shown that $\Sigma_2^p \subseteq \mathrm{RP}^{\mathrm{NP}}$.

It follows that $\Sigma_2^p \subseteq \mathrm{BPP}^{\mathrm{NP}}$. This is known to imply $\mathrm{PH} = \mathrm{BPP}^{\mathrm{NP}}$. □

The above theorem leads to an interesting remark. By Cai's algorithm [6], we can determine in $\mathrm{ZPP}^{\mathrm{NP}}$ whether an IC is found on the row-side or the column-side, but we cannot find an IC in $\mathrm{ZPP}^{\mathrm{NP}}$. The above theorem also rules out the possibility of designing a $\mathrm{FP}^{\mathrm{prAM}}$ for finding irrefutable certificates. However, we show that *collectively irrefutable certificates* (written CIC) can be found in $\mathrm{FP}^{\mathrm{prAM}}$.

**Definition 4.6** *Let $M$ be a $2^n \times 2^m$ Boolean matrix. A set of rows $Y \subseteq \{0,1\}^n$ 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 $Z \subseteq \{0,1\}^m$ is called a column-side* CIC*, if for every row $y$, there exists a column $z \in Z$ such that $z$ beats $y$.*

In the next section, we shall present a $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding a CIC and prove the following theorem. The algorithm takes an input a row-side $\mathrm{S}_2$-type matrix and outputs a row-side CIC. A similar algorithm for the column-side is also presented.

**Theorem 4.7** *There exists a* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes as input a circuit succinctly representing a row-side* $\mathrm{S}_2$*-type matrix (respectively, a column-side* $\mathrm{S}_2$*-type matrix)* $M$ *of size* $2^n \times 2^m$*, and outputs a row-side* CIC *of size* $m$ *(respectively, a column-side* CIC *of size* $n$*).*

Using the above theorem, it is easy to construct an algorithm that takes as input any $\mathrm{S}_2$-type matrix and outputs a CIC, either on the row-side or the column-side.

**Theorem 4.8** *There exists a* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes as input a circuit succinctly representing a* $\mathrm{S}_2$*-type matrix* $M$ *of size* $2^n \times 2^m$ *and outputs either a row-side* CIC *of size* $m$ *or a column-side* CIC *of size* $n$*.*

*Proof:* The proof is a simple application of Theorem 4.7, which gives us two algorithms: (i) an algorithm that takes as input a row-side $\mathrm{S}_2$-type matrix and outputs a row-side CIC; (ii) an algorithm that takes as input a column-side $\mathrm{S}_2$-type matrix and outputs a column-side CIC. We run both these algorithms on the input matrix $M$. The given matrix $M$ is guaranteed to be either a row-side $\mathrm{S}_2$-type matrix or a column-side $\mathrm{S}_2$-type matrix. Thus, one of the two runs would output a CIC. The other run would output some arbitrary result, since 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. The above check can be performed by making a single NP query. □

To summarize, we cannot design a $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding an IC of a given $\mathrm{S}_2$-type matrix, but we can indeed find a CIC of an input $\mathrm{S}_2$-type matrix in $\mathrm{FP}^{\mathrm{prAM}}$. Theorems 4.7 and Theorem 4.8 yield a few corollaries, discussed in the subsections below.

## 4.1 Upperbounds for $\mathrm{S}_2^p$

In this section, we show that $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$ and also derive an alternative of Cai's result [6] that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

**Theorem 4.9** $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$.

*Proof:* The claim follows directly from Theorem 4.8. Let $L$ be a language in $\mathrm{S}_2^p$. Let $x$ be the input string. Consider the $\mathrm{S}_2$-type matrix $M$ corresponding to $x$. We can obtain a circuit $C$ succinctly representing the matrix $M$ in time polynomial in $|x|$. Invoking the algorithm given in Theorem 4.8 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 $\mathrm{P}^{\mathrm{prAM}}$ is. We observe that $\mathrm{P}^{\mathrm{prAM}} \subseteq \mathrm{BPP}^{\mathrm{NP}}$. The containment relationships between $\mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{P}^{\mathrm{prAM}}$ are unknown. So, our result that $\mathrm{S}_2^p \subseteq \mathrm{P}^{\mathrm{prAM}}$ is incomparable to Cai's result [6] that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. Here, we observe that an alternative proof of Cai's result can be derived using Theorem 4.8.

**Theorem 4.10 ([6])** $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

*Proof:* Consider a language $L \in \mathrm{S}_2^p$ and we shall describe a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for deciding $L$. Given an input $x$, we first construct a circuit $C$ succinctly representing the $\mathrm{S}_2$-type matrix $M$ corresponding to $x$. We next invoke the algorithm given in Theorem 4.8 with $C$ as the input. Whenever a prAM oracle query $q$ is issued by the algorithm, we simulate the prAM protocol on $q$ by making use of the NP oracle and by tossing coins. With high probability, the simulation yields the correct answer for $q$. Continuing this way, we obtain an output $s$. We would expect $s$ to be a row-side `CIC` or a column-side `CIC`. But, it is possible that $s$ is not a `CIC`, because of the error in our simulation of the prAM protocol. However, this event occurs with a low probability. The output of our procedure is as follows: if $s$ is a row-side `CIC`, output "$x \in L$"; if $s$ is a column-side `CIC`, output "$x \notin L$"; if both the tests fail, output "?". The above tests are performed by making queries to the NP oracle.

Since $M$ is an $\mathrm{S}_2$-type matrix, the presence of a row-side `CIC` implies that $x \in L$. Similarly, the presence of a column-side `CIC` implies that $x \notin L$. This means that our procedure has zero-error. We already observed that the procedure has high probability of success. Thus, we have exhibited a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for $L$. □

## 4.2 Consequences of NP having small circuits

A body of prior work has dealt with the implications 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 [18] showed that, under this assumption, the polynomial time hierarchy (PH) collapses to $\Sigma_2^p \cap \Pi_2^p$, i.e., $\mathrm{PH} = \Sigma_2^p \cap \Pi_2^p$. Köbler and Watanabe [20] improved the collapse to $\mathrm{ZPP}^{\mathrm{NP}}$. Sengupta (see [6]) observed that the collapse can be brought down to $\mathrm{S}_2^p$. This has been further improved via a collapse to $\mathrm{O}_2^p$, the oblivious version of $\mathrm{S}_2^p$ [8]. It has been an interesting open problem to obtain a collapse to the class $\mathrm{P}^{\mathrm{NP}}$. Here, we show a collapse to $\mathrm{P}^{\mathrm{prMA}}$.

**Theorem 4.11** *If* $\mathrm{NP} \subseteq \mathrm{P/poly}$*, then* $\mathrm{PH} = \mathrm{P}^{\mathrm{prMA}}$.

*Proof:* By Sengupta's observation (see [6]), the assumption implies that $\mathrm{PH} = \mathrm{S}_2^p$. Combining this with Theorem 4.9, we get $\mathrm{PH} = \mathrm{P}^{\mathrm{prAM}}$. Arvind et al. [3] showed that if $\mathrm{NP} \subseteq \mathrm{P/poly}$ then $\mathrm{AM} = \mathrm{MA}$. We observe that this result carries over to the promise versions, namely the same assumption implies $\mathrm{prAM} = \mathrm{prMA}$. 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. While we can show that $\mathrm{P}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ (see Appendix), it is open whether $\mathrm{P}^{\mathrm{prMA}}$ is contained in $\mathrm{S}_2^p$.

Under the assumption NP has polynomial size circuits, Bshouty et. al [5] studied the problem of learning a correct circuit for SAT and designed a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm. As an application of Theorem 4.7, we obtain an $\mathrm{FP}^{\mathrm{prMA}}$ algorithm for the same problem. We can show that $\mathrm{FP}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ (see Appendix) and thus, the new result provides an improvement for this problem. The proof uses a construction similar to the one used by Fortnow et. al [13].

**Theorem 4.12** *If* $\mathrm{NP} \subseteq \mathrm{P/poly}$*, then there exists an* $\mathrm{FP}^{\mathrm{prMA}}$ *algorithm that takes as input a number* $n$ *(in unary) and outputs a correct circuit for* SAT *at length* $n$.

*Proof:* Our assumption implies that SAT is computed by circuits of size $n^k$, for some constant $k$.

Let $C$ be a (possibly incorrect) circuit claimed to compute SAT at a certain length $n$. We say that $C$ is *nice*, if $C$ does not accept unsatisfiable formulas. It is well-known that the circuit $C$ can be converted into a nice circuit $C'$ while preserving correctness; namely, if $C$ is a correct then $C'$ is also a correct circuit. The above transformation goes via self-reducibility and it can be performed in polynomial time.

We shall first describe an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding a correct circuit for SAT at the given length $n$. Define a matrix $M$ as follows. Each circuit of size $n^k$ is a row in this matrix. Each column in $M$ corresponds to a pair $\langle \varphi, t \rangle$, where $\varphi$ is a formula of length $n$ and $t$ is a truth assignment for $\varphi$. The length of the pair is $2n$. Altogether the matrix $M$ is of size $2^{n^k} \times 2^{2n}$. The entry $M[C, \langle \varphi, t \rangle]$ is given by the following procedure. Convert $C$ into a nice circuit $C'$ and consider the two cases:

- Case 1: Suppose $\varphi(t) = \mathit{true}$. If $C'(\varphi) = 1$, set the entry to 1, else set it to 0.
- Case 2: Suppose $\varphi(t) = \mathit{false}$. Set the entry to 1.

By our assumption, there exists a circuit $C^*$ of size $n^k$ that correctly computes SAT at length $n$. Notice that the row in $M$ corresponding to $C^*$ is full of 1's and hence, $M$ is an $\mathrm{S}_2$-type matrix having a row-side `IC`. Moreover, we can construct a circuit succinctly representing the matrix $M$ in time polynomial in $n$.

Invoke the algorithm given in Theorem 4.7 and obtain row-side `CIC` $\mathcal{C}$ of cardinality $2n$. The set $\mathcal{C}$ consists of $2n$ circuits, each of size $n^k$. Convert each circuit $C \in \mathcal{C}$ in to a nice circuit $C'$. Let $\mathcal{C}'$ denote the collection of these nice circuits. We make an observation regarding $\mathcal{C}'$.

Let $\varphi$ be any formula of length $n$ and consider the following two cases:

- $\varphi$ is unsatisfiable: In this case, every circuit $C' \in \mathcal{C}'$ rejects $\varphi$, because $C'$ is a nice circuit.
- $\varphi$ is satisfiable: Let $t$ be a satisfying truth assignment of $\varphi$. Since $\mathcal{C}$ is a `CIC`, some circuit $C \in \mathcal{C}$ beats $\langle \varphi, t \rangle$. This means that the nice circuit $C'$ corresponding to $C$ accepts $\varphi$.

To summarize, any unsatisfiable formula is rejected by all the circuits in $\mathcal{C}'$ and any satisfiable formula is accepted by at least one of the circuits in $\mathcal{C}'$.

Based on the above observation, we can construct a correct circuit $\widetilde{C}$ for SAT at length $n$. We simply take $\widetilde{C}$ to be the disjunction of all the circuits in $\mathcal{C}'$, i.e., $\widetilde{C}$ accepts a given formula $\varphi$, if and only if some circuit $C' \in \mathcal{C}'$ accepts $\varphi$. An easy calculation shows that $\widetilde{C}$ is of size $O(n^{k+2})$.

The above description gives an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding circuits for SAT. The theorem now follows from a result due to Arvind et al. [3]: if $\mathrm{NP} \subseteq \mathrm{P/poly}$ then $\mathrm{AM} = \mathrm{MA}$. We observe that this result carries over to the promise versions, namely the same assumption implies $\mathrm{prAM} = \mathrm{prMA}$. □

As a corollary of the above theorem, we can design a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for learning circuits for SAT, assuming NP has polynomial size circuits. This provides an alternative proof of the same result by Bshouty et al. [5].

**Theorem 4.13** *If* $\mathrm{NP} \subseteq \mathrm{P/poly}$*, then there exists an* $\mathrm{ZPP}^{\mathrm{NP}}$ *algorithm that takes as input a number* $n$ *(in unary) and outputs a correct circuit for* SAT *at length* $n$*.*

*Proof:* We simulate the $\mathrm{P}^{\mathrm{prAM}}$ algorithm given in Theorem 4.12. Whenever a query $q$ is asked, we simulate the prAM protocol by tossing coins and making use of the NP oracle. At the end, we have a circuit $C$. We convert $C$ into a nice circuit $C'$. It is guaranteed that $C'$ does not accept unsatisfiable formulas. We can make a query to the NP oracle to check if $C'$ rejects any satisfiable formulas. Thus, we can verify if $C'$ is a correct circuit. If so, we output $C'$; else output "?". □

### 4.3 Collectively Irrefutable Certificates for Arbitrary Boolean Matrices

Theorem 4.8 deals only with $S_2$-type matrices. In this section, we describe a generalization that handles arbitrary Boolean matrices.

Any $S_2$-type matrix has either a row-side CIC or a column-side CIC, but not both. But, in the case of arbitrary Boolean matrices, both a row-side CIC and a column-side CIC may exist. Goldreich and Wigderson [15] proved the following combinatorial result that asserts the existence of a small CIC in any Boolean matrix. We rephrase their result (Lemma 6 in [15]) using our terminology:

**Theorem 4.14 ([15])** *Let $M$ be any $2^n \times 2^m$ Boolean matrix. At least one of the following statements is true: (i) there exists a row-side* CIC *of size $m$ (i.e., the log of the number of columns); (ii) there exists a column-side* CIC *of size $n$ (i.e., the log of the number of rows).*

We obtain the following constructive version of the above result (with a slight blow-up in the size of the CIC). For this, we will apply Theorem 4.7 and Theorem 4.14.

**Theorem 4.15** *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes as input a circuit $C$ succinctly representing a $2^n \times 2^m$ Boolean matrix $M$, and outputs either a row-side* CIC *of size $m^2$ or a column-side* CIC *of size $n^2$.*

*Proof:* We define two Boolean matrices $\overline{M}_1$ and $\overline{M}_2$ as follows.

The matrix $\overline{M}_1$ is of size $2^{nm} \times 2^m$. Each row of $\overline{M}_1$ corresponds to a sequence $\langle y_1, y_2, \ldots, y_m \rangle$ of $m$ rows of $M$. Each column $z$ of $\overline{M}_1$ corresponds to a single column of $M$. The entries of $\overline{M}_1$ are defined as below. For a row $\langle y_1, y_2, \ldots, y_m \rangle \in \{0,1\}^{nm}$ and a column $z \in \{0,1\}^m$, the entry is defined as:

$$\overline{M}_1[\langle y_1, y_2, \ldots, y_m \rangle, z] \quad = \quad \begin{cases} 1 & \text{if some } y_i \text{ beats } z \text{ in } M \\ 0 & \text{otherwise} \end{cases}$$

The matrix $\overline{M}_2$ is defined analogously. Each row $y$ of $\overline{M}_1$ corresponds to a single row of $M$. Each column of $\overline{M}_2$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_n \rangle$ of $n$ columns of $M$. Thus, $\overline{M}_2$ is matrix of size $2^n \times 2^{mn}$. The entries of $\overline{M}_2$ are defined as below. For a row $y$ and a column $\langle z_1, z_2, \ldots, z_n \rangle \in \{0,1\}^{mn}$, the entry is defined as:

$$\overline{M}_2[y, \langle z_1, z_2, \ldots, z_n \rangle] \quad = \quad \begin{cases} 0 & \text{if some } z_i \text{ beats } y \text{ in } M \\ 1 & \text{otherwise} \end{cases}$$

Using Theorem 4.14, we observe that at least one of the following two claims is true: (i) $\overline{M}_1$ is an $S_2$-type matrix having a row-side IC; (ii) $\overline{M}_2$ is an $S_2$-type matrix having a column-side IC. The first scenario occurs, if $M$ has a row-side CIC of size $m$ and the second scenario occurs, if $M$ has a column-side CIC of size $n$.

We can construct in polynomial time circuits $\overline{C}_1$ and $\overline{C}_2$ that succinctly encode the matrices $\overline{M}_1$ and $\overline{M}_2$, respectively. We run the algorithm given in Theorem 4.7 on both the matrices. Let $S_1$ and $S_2$ denote the output of the two runs. The above algorithm requires that the input be an $S_2$-type matrix. We satisfy this requirement in at least one of the two runs. Thus, at least one of the following claims is true: (i) $S_1$ is a row-side CIC of size $m$ for $\overline{M}_1$; (ii) $S_2$ is a column-side CIC of size $n$ for $\overline{M}_2$. We can check whether a given set is a row-side (respectively, column-side) CIC by making a single query to an NP oracle. So, these tests can certainly be performed using a prAM

oracle. Among the two sets $S_1$ and $S_2$, we choose a set that passes the above test. Suppose $S_1$ is chosen. Then, $S_1$ is a CIC for $\overline{M}_1$ of size $m$. Each row $\overline{y} \in S_1$ is a collection of $m$ rows of $M$. By taking the union of these collections over all $\overline{y} \in S_1$, we get a row-side CIC of size $m^2$ for $M$. On the other hand, if $S_2$ is chosen, a similar process produces a column-side CIC of size $n^2$ for $M$. □

The above theorem provides an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding a small CIC for arbitrary Boolean matrices. We note that the same task can also be accomplished in $\mathrm{ZPP}^{\mathrm{NP}}$; this can be shown via an argument similar to that of Theorem 4.10.

# 5 Finding a CIC: Proof of Theorem 4.7

In this section, we prove Theorem 4.7. We shall only describe the algorithm for the case of row-side $\mathrm{S}_2$-type matrices. The case of column-side $\mathrm{S}_2$-type matrices is handled in a similar manner.

The algorithm computes the required row-side CIC using a simple 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 5.1, 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 $Q$. In each iteration, we invoke the algorithm by setting $Q = U_k$. 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 representing 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 Theorem 4.7, modulo Lemma 5.1.

## 5.1 Finding a Good Row

This section is devoted to proving the following lemma used in the previous section.

**Lemma 5.1** *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that solves the following problem. The input consists of: (i) a circuit succinctly representing a row-side* $\mathrm{S}_2$*-type matrix* $M$ *of dimension* $2^n \times 2^m$*; (ii) a set of columns* $Q \subseteq \{0,1\}^m$ *presented succinctly in the form of a circuit; The output is a row* $\widehat{y} \in \{0,1\}^n$ *that beats at least half the columns in* $Q$*.*

We build the required string $\widehat{y}$ 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 row $\widehat{y}$ by performing self-reduction that incurs a small amount of "loss" in the "goodness" in each iteration. We formalize the notion of goodness and then describe the algorithm.

**Definition 5.2** *Let $A$ be a $2^a \times 2^b$ Boolean matrix. For a row $y \in \{0,1\}^a$ and a set of columns $X \subseteq \{0,1\}^b$, we define* goodness

$$\mu(y, X) = \frac{|\{z \in X \ : \ y \text{ beats } z\}|}{|X|}$$

*We extend the notion of goodness to prefixes. Let $s$ be a string of length $|s| \leq a$. We say that a row $y \in \{0,1\}^a$ is an* extension *of $s$, if $s$ is a prefix of $y$. The* goodness *of the prefix $s$ with respect to $X$ is defined to be the maximum goodness we can achieve by extending $s$ into a full row:*

$$\mu(s, X) = \max_{y \in \{0,1\}^a \ : \ y \text{ extends } s} [\mu(y, X)]$$

We now describe the algorithm claimed in Lemma 5.1. Our goal is to find a row $\widehat{y}$ such that $\mu(\widehat{y}, Q) \geq 1/2$. The algorithm starts with a prefix $s_0 = \lambda$, where $\lambda$ is the empty string. It constructs the required row $\widehat{y}$ in $n$ iterations, adding one bit in each iteration. In the $k^{th}$ iteration, it has a prefix $s_{k-1}$ from the previous iteration; it finds a suitable bit $b_k \in \{0,1\}$ and appends it to $s_{k-1}$ to get a new prefix $s_k = s_{k-1}b_k$. Continuing this way for $n$ iterations, the algorithm finds a string $s_n$ of length $n$ and outputs $\widehat{y} = s_n$. Throughout the process, the algorithm would ensure that the prefixes constructed have certain amount of goodness.

We have assumed that the given matrix $M$ is a row-side $S_2$-type matrix and so, it has a row-side `IC` $y^*$. Notice that $\mu(s_0, Q) = 1$, because $y^*$ is an extension of (the empty string) $s_0$ and $y^*$ beats every column in $M$. Thus, the algorithm has been started off with a prefix $s_0$ having goodness $\mu(s_0, Q) = 1$. We shall ensure that the algorithm loses at most $\epsilon$ amount of goodness in each iteration ($\epsilon$ is a parameter fixed below). We shall ensure inductively that for $0 \leq k \leq n$, $\mu(s_k, Q) \geq 1 - k\epsilon$.

We now need to describe how to find a suitable bit in each iteration. Consider $k^{th}$ iteration, for some $k \geq 1$. We have a prefix $s_{k-1}$ from the previous iteration. Write $s = s_{k-1}$ and $\rho = 1-(k-1)\epsilon$. By induction, we can assume that $\mu(s, Q) \geq \rho$. Our aim is to find a bit $b$ such that $\mu(sb, Q) \geq \rho - \epsilon$. The main observation is that $\mu(s, Q) = \max\{\mu(s0, Q), \mu(s1, Q)\}$. Thus, at least one of the following is true: $\mu(s0, Q) \geq \rho$ or $\mu(s1, Q) \geq \rho$.

The bit $b$ is found by invoking the algorithm described in Lemma 5.3: given a prefix $\beta$ the algorithm can distinguish between the cases of $\mu(\beta, Q) \geq \rho$ and $\mu(\beta, Q) \leq \rho - \epsilon$. We invoke algorithm two times with $\beta = s0$ and $\beta = s1$ as inputs. By the above observation, the algorithm should answer "yes" on at least one of these invocations. Let $b$ be a bit such that the algorithm answers "yes" on input $\beta = sb$. Observe that $\mu(sb, Q) \geq \rho - \epsilon$ (for otherwise, the algorithm would have answered "no" on input $\beta = sb$).

By the above procedure, we can find a row $\widehat{y}$ such that $\mu(\widehat{y}, Q) \geq 1 - n\epsilon$. Setting $\epsilon = 1/n^2$, we see that $\widehat{y}$ beats at least a fraction of $(1 - 1/n) \geq 1/2$ columns in $Q$. We have proved Lemma 5.1, modulo Lemma 5.3.

## 5.2 Testing for the Goodness of a Prefix

In this section, we prove the following lemma used in the previous section. The lemma tests whether a prefix is good.

**Lemma 5.3** *There exists an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm that solves the following problem. The input consists of: (i) a circuit succinctly representing a row-sided $S_2$-type matrix $M$ of dimension $2^n \times 2^m$;*

*(ii) a circuit succinctly representing a set of columns $Q \subseteq \{0,1\}^m$ (iii) a string $\beta$ having length $|\beta| \leq n$; (iv) two parameters $\rho \leq 1$ and $\epsilon > 0$. It outputs an answer as follows:*

$$\begin{aligned} \mu(\beta, Q) \geq \rho &\implies \textit{Answer}=\text{“yes”} \\ \mu(\beta, Q) \leq \rho - \epsilon &\implies \textit{Answer}=\text{“no”}. \end{aligned}$$

*If $\mu(\beta, Q)$ falls in the interval $[\rho - \epsilon, \rho]$, the output can be arbitrary. The running time of the algorithm has a polynomial dependence on $1/\epsilon$ and $1/\rho$.*

The gap between the positive and negative instances in the above problem is only $\epsilon$. We shall first amplify the gap via the standard trick of running repeated trials and applying Chernoff bounds.

The amplification involves a parameter $r$, to be fixed shortly. Define a new matrix $\overline{M}$ as follows. Each row in $\overline{M}$ corresponds to a row in $M$ and each column of $\overline{M}$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_r \rangle$ of $r$ columns from $M$. Thus, $\overline{M}$ is of dimension $2^n \times 2^{\overline{m}}$, where $\overline{m} = mr$. For a row $y \in \{0,1\}^n$ and a column $\overline{z} = \langle z_1, z_2, \ldots z_r \rangle$, define the entry

$$\overline{M}[y, \overline{z}] \;=\; \begin{cases} 1 & \text{if } \dfrac{|\{z_i : y \text{ beats } z_i \text{ in } M\}|}{r} \geq (\rho - \frac{\epsilon}{2}) \\ 0 & \text{otherwise} \end{cases}$$

For a set of columns $X \subseteq \{0,1\}^m$, define $X^r$ to be the $r$-way Cartesian product of $X$:

$$X^r = \{\langle z_1, z_2, \ldots, z_r \rangle \;:\; \text{for } 1 \leq j \leq r,\, z_i \in X\}.$$

Notice that $X^r$ is a subset of columns of $\overline{M}$.

Set $r = \lceil 16m/\epsilon^2 \rceil$. Applying Chernoff bounds, we get the following claim, which expands the gap from $[\rho - \epsilon, \rho]$ to $[1/\overline{m}^4, 1/2]$.

**Lemma 5.4** *Consider a row $y \in \{0,1\}^n$. Let $\overline{Q} = Q^r$ be the corresponding set of columns in the matrix $\overline{M}$. Then,*

$$\begin{aligned} \mu(y, Q) \geq \rho &\implies \mu(y, \overline{Q}) \geq 1/2 \\ \mu(y, Q) \leq \rho - \epsilon &\implies \mu(y, \overline{Q}) \leq 1/\overline{m}^4 \end{aligned}$$

The above lemma enables us to focus on $\overline{M}$ and $\overline{Q}$, instead of $M$ and $Q$. Our aim is to test whether the given prefix $\beta$ has an extension $y$ that beats a sufficiently large *fraction* of columns from $\overline{Q}$. This is accomplished in two successive steps. First, we estimate the size of the set $\overline{Q}$. In the second step, we test whether $\beta$ has an extension $y$ that beats a sufficiently large *number* of columns from the set $\overline{Q}$. The first step involves approximate counting and this is accomplished using Theorem 3.2. The second step involves testing whether a given set is large, which we will perform by making oracle queries to a promise language. A lemma due to Sipser [30] is used for proving that this promise language indeed lies in the class prAM. The following notation is needed for stating 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).

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

**Lemma 5.5 ([30])** *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 $\Pr_H[S \text{ has a collision under } H] \leq 1/2$.*

Our algorithm would make oracle queries to the following promise language.

**Promise Language** GoodPrefix: The instances of this language consist of (i) a circuit succinctly representing a $2^n \times 2^m$ matrix $M$; (ii) a string $\beta$ having length $|\beta| \leq n$; (iii) a circuit succinctly representing a set of columns $X \subseteq \{0,1\}^m$, where each set $X_i \subseteq \{0,1\}^m$ is presented succinctly in the form of a circuit; (iv) a positive integer $k$.
*Positive instances:* There exists a row $y$ extending $\beta$ such that $y$ beats at least $k2^k$ columns in $X$.
*Negative instances:* For all rows $y$ extending $\beta$, $y$ beats at most $2^k$ columns in $X$.

The following lemma shows that GoodPrefix lies in the promise class prAM.

**Lemma 5.6** *The promise language* GoodPrefix *belongs to the promise class* prAM.

*Proof:* We present an MAM protocol; it is well known such a protocol can be converted into an AM protocol [4].

Merlin claims that a given instance is of the positive type. To prove this, he provides a row $y$ extending $\beta$. 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 $|Z| \leq 2^{k-1}$. 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 then prove that $Z$ has a collision under $H$. Arthur accepts, if Merlin can prove this; otherwise, Arthur rejects. The correctness of the protocol follows from Lemma 5.5. □

We now describe the algorithm claimed in Lemma 5.3. We invoke the algorithm given in Theorem 3.2 on the set $\overline{Q}$ and find a estimate $U$ such that $(1-\widehat{\delta})U \leq |\overline{Q}| \leq U$, where $\widehat{\delta}$ is a parameter suitably set as $\widehat{\delta} = 0.1$. Fix $k = \lfloor \log \frac{(1-\widehat{\delta})U}{2\overline{m}} \rfloor$. The choice of $k$ ensures that for any row $y$

$$\begin{aligned} \mu(y, \overline{Q}) \geq 1/2 &\implies y \text{ beats at least } k2^k \text{ columns in } \overline{Q} \\ \mu(y, \overline{Q}) \leq 1/\overline{m}^4 &\implies y \text{ beats at most } 2^{k-1} \text{ columns in } \overline{Q} \end{aligned}$$

We ask the GoodPrefix oracle a single query consisting of $\overline{M}$, $\beta$, $\overline{Q}$ and $k$. If the oracle answers "yes", we output "yes"; otherwise, we output "no". The correctness of the algorithm follows from Lemma 5.4.

# 6 Finding Strong Collectively Irrefutable Certificates

In this section, we extend the notion of `CIC` to *Strong* `CIC` and present an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding these objects. This algorithm is used in the next section for computing near-optimal strategies of succinct zero-sum games. We believe that the notion of strong `CIC` and the algorithm developed here may be of independent interest.

**Definition 6.1** *Let $M$ be a $2^n \times 2^m$ Boolean matrix. For $0 \leq \alpha \leq 1$, a set of rows $Y \subseteq \{0,1\}^n$ is called a row-side $\alpha$-strong* CIC*, if for any column $z$, at least $\alpha$ fraction of rows from $Y$ beat $z$, i.e., $|\{y \in Y \ : \ y \text{ beats } z\}| \geq \alpha|Y|$. Similarly, a set of columns $Z \subseteq \{0,1\}^m$ is called a column-side $\alpha$-strong* CIC*, if for any column $y$, at least $\alpha$ fraction of rows from $Z$ beat $y$, i.e., $|\{z \in Z \ : \ z \text{ beats } y\}| \geq \alpha|Z|$.*

A $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for finding $\alpha$-strong CIC of a given row-side $\mathrm{S}_2$-type matrix is implicit in the work of Cai ([6], Section 5) [5]:

**Theorem 6.2 ([6])** *There exists a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm that takes an input a circuit succinctly representing a row-side $\mathrm{S}_2$-type matrix (respectively, a column-side $\mathrm{S}_2$-type matrix) $M$ of size $2^n \times 2^m$ and a parameter $\alpha < 1$, and outputs a row-side $\alpha$-strong* CIC *(respectively, a column-side* CIC*) of size polynomial in $1/(1-\alpha)$, $n$ and $m$. The running time of the algorithm is polynomial in the size of the circuit and $1/(1-\alpha)$.*

Our main result here is a $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for the same problem considered in the above theorem.

**Theorem 6.3** *There exists a $\mathrm{FP}^{\mathrm{prAM}}$ algorithm that takes an input a circuit succinctly representing a row-side $\mathrm{S}_2$-type matrix (respectively, a column-side $\mathrm{S}_2$-type matrix) $M$ of size $2^n \times 2^m$ and a parameter $\alpha < 1$, and outputs a row-side $\alpha$-strong* CIC *(respectively, a column-side* CIC*) of size polynomial in $1/(1-\alpha)$, $n$ and $m$. The running time of the algorithm is polynomial in the size of the circuit and $1/(1-\alpha)$.*

The rest of the section is devoted to proving the above theorem. We shall only discuss the case of row-side $\mathrm{S}_2$-type matrices. The other case of column-side $\mathrm{S}_2$-type matrices is handled in a similar manner.

We obtain the $\mathrm{FP}^{\mathrm{prAM}}$ algorithm by combining and extending the ideas from Cai's algorithm (Theorem 6.2) and our algorithm for finding CIC (Theorem 4.7). Our $\mathrm{FP}^{\mathrm{prAM}}$ algorithm follows Cai's algorithm closely; it differs from his algorithm in a sub-procedure used for performing a crucial task. Cai presents a $\mathrm{ZPP}^{\mathrm{NP}}$ procedure for performing the task, while we present a $\mathrm{FP}^{\mathrm{prAM}}$ procedure. Accordingly, we split the proof of the theorem into two parts. We first describe the overall algorithm. The second part presents our $\mathrm{FP}^{\mathrm{prAM}}$ procedure that performs the above mentioned task.

## 6.1 Overall Algorithm

The algorithm uses two parameters $p < 1$ and $t$ suitably fixed as $p = (1+\alpha)/2$ and $t = 2(m+1)(1+\alpha)/(1-\alpha)$. The algorithm constructs a row-side $\alpha$-strong CIC $Y = \{y_1, y_2, \ldots, y_t\}$ iteratively by choosing a suitable row $y_k$ in the $k^{th}$ iteration.

Let $Z_0^0 = \{0,1\}^m$ denote the set of all columns. The row $y_1$ is chosen to be a row that beats at least $p$ fraction of the columns in $Z_0^0$. Suppose we have chosen the rows $y_1, y_2, \ldots, y_k$ and consider the $(k+1)st$ iteration, where the row $y_{k+1}$ is to be chosen. Partition the set $\{0,1\}^m$ into disjoint

[5] The algorithm is actually given for the special case of $\alpha = 1/2$; however, it can easily be adapted to handle the case of larger $\alpha$

sets $Z_0^k, Z_1^k, \ldots, Z_k^k$ as follows. For $0 \leq d \leq k$, let $Z_d^k$ be the set of all columns that are beaten by exactly $d$ rows from $y_1, y_2, \ldots, y_k$, i.e., $Z_d^k$ is the set of all columns $z \in \{0,1\}^m$ such that

$$|\{y_i \ : \ 1 \leq i \leq k, y_i \text{ beats z}\}| \ = \ d$$

We choose a row $y_{k+1}$ such that it beats at least $p$ fraction of the columns in each $Z_d^k$:

$$\text{for all } 0 \leq d \leq k, \quad |\{z \in Z_d^k \ : \ y_{k+1} \text{ beats } z\}| \geq p \cdot |Z_d^k| \tag{1}$$

Notice that a row-side IC meets the above constraint, since it beats every column in $\{0,1\}^m$. Hence, our assumption that $M$ is a row-side $S_2$-type matrix implies the existence of a row meeting the required constraint. However, our algorithm does not have a handle on the row-side IC.

We obtain a row $y_{k+1}$ satisfying the above constraint (1) by invoking the $\text{FP}^{\text{prAM}}$ procedure presented in Lemma 6.6. The procedure is invoked by setting $\mathcal{Q} = \{Z_0^k, Z_1^k, \ldots, Z_k^k\}$.

At the end of $t$ iterations, the algorithm has found rows $y_1, y_2, \ldots y_t$ meeting the constraints and outputs the set $Y = \{y_1, y_2, \ldots, y_t\}$. The main claim is that the set $Y$ is a row-side $\alpha$-strong CIC. The proof of the claim is outlined below.

For $0 \leq k \leq t$ and $0 \leq d \leq k$, partition the set $Z_d^k$ into two disjoint sets $Z_d^k(0)$ and $Z_d^k(1)$, where

$$\begin{aligned} Z_d^k(0) &= \{z \in Z_d^k \ : \ y_{k+1} \text{ does not beat } z\} \\ Z_d^k(1) &= \{z \in Z_d^k \ : \ y_{k+1} \text{ beats } z\} \end{aligned}$$

Notice that $Z_d^{k+1} = Z_d^k(0) \cup Z_{d-1}^k(1)$. Informally, we visualize that the row $y_{k+1}$ splits each set $Z_d^k$ into $Z_d^k(0)$ and $Z_d^k(1)$ and sends them to $Z_d^{k+1}$ and $Z_{d+1}^{k+1}$, respectively. The main constraint satisfied by $y_{k+1}$ is that $|Z_d^k(1)| \geq p|Z_d^k|$. Similarly, we visualize that the set $Z_d^{k+1}$ receives its contents from $Z_d^k$ and $Z_{d-1}^k$. The concept is illustrated in Figure 1.

With these notations, the main claim that $Y$ is a row-side $\alpha$-strong CIC can be stated equivalently as: for $0 \leq d \leq \lceil \alpha t \rceil$, the set $Z_d^t$ is empty. The claim is implied by the following combinatorial lemma due to Cai [6]. (There the lemma is proved for the specific case of $\alpha = 1/2$; we obtain the generalization to higher values of $\alpha$ by suitably setting the parameters $p$ and $t$ based on the input $\alpha$).

**Lemma 6.4** *[6] Let $Z^0, Z^1, \ldots, Z^t$ be a sequence of partitions of $\{0,1\}^m$, where each $Z^k$ partitions the set $\{0,1\}^m$ into disjoint sets $Z_0^k, Z_1^k, \ldots, Z_k^k$. Divide each set $Z_d^k$ into two disjoint sets $Z_d^k(0)$ and $Z_d^k(1)$ such that the following constraints are satisfied: (i) $Z_d^{k+1} = Z_d^k(0) \cup Z_{d-1}^k(1)$; (ii) $Z_d^k(1) \geq p|Z_d^k|$. Then,*

$$Z_0^t = Z_1^t = \cdots = Z_{\lceil \alpha t \rceil}^t = \emptyset,$$

*provided $p = \frac{1+\alpha}{2}$ and $t \geq \frac{2(m+1)(1+\alpha)}{1-\alpha}$.*

## 6.2 Finding a Good Row

In this section, we prove Lemma 6.6 used in the previous section. We first extend the notion of goodness (Definition 5.2) to handle a collection of sets of columns.

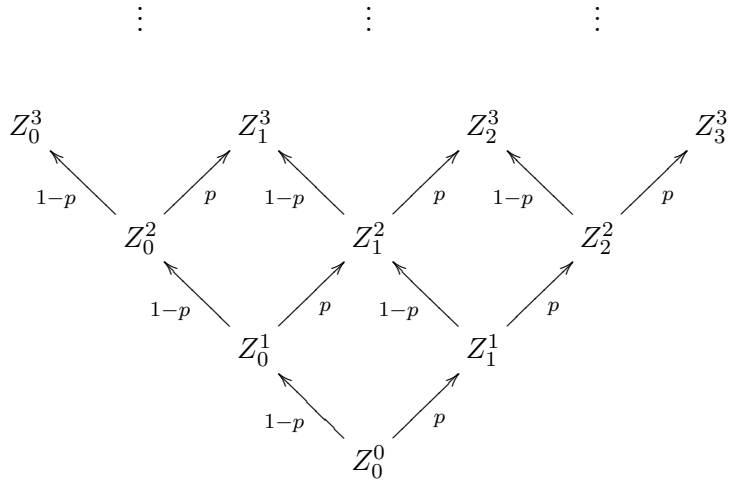


Figure 1: Illustration for Lemma 6.4

**Definition 6.5** *Let $A$ be a $2^a \times 2^b$ Boolean matrix. For a row $y \in \{0,1\}^a$ and a set of columns $X \subseteq \{0,1\}^b$, we define*

$$\mu(y, X) = \frac{|\{z \in X \ : \ y \text{ beats } z\}|}{|X|}$$

*Consider a set $\mathcal{X} = \{X_1, X_2, \ldots, X_\ell\}$, where each $X_i \subseteq \{0,1\}^a$ is a subset of columns of $A$. The* goodness *of a row $y$ with respect to $\mathcal{X}$ is defined as:*

$$\mu(y, \mathcal{X}) = \min_{i=1}^{\ell} \mu(y, X_i).$$

*We extend the notion of goodness to prefixes. Let $s$ be a string of length $|s| \leq a$. We say that a row $y \in \{0,1\}^a$ is an* extension *of $s$, if $s$ is a prefix of $y$. The* goodness *of the prefix $s$ with respect to $\mathcal{X}$ is defined to be the maximum goodness we can achieve by extending $s$ into a full row:*

$$\mu(s, \mathcal{X}) = \max_{y \in \{0,1\}^a \ : \ y \text{ extends } s} [\mu(y, \mathcal{X})]$$

**Lemma 6.6** *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that solves the following problem. The input consists of: (i) a circuit succinctly representing a row-side* $\mathrm{S}_2$*-type matrix $M$ of dimension $2^n \times 2^m$; (ii) a set $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_\ell\}$, where each $Q_i \subseteq \{0,1\}^m$ is a subset of the columns of $M$ that is presented succinctly in the form of a circuit; (iii) a parameter $p < 1$. The output is a row $\widehat{y}$ having goodness $\mu(\widehat{y}, \mathcal{Q}) \geq p$. The running time of the algorithm has a polynomial dependence on $1/(1-p)$.*

Notice that the above lemma is a generalization of Lemma 5.1 that can handle a collection of sets of columns. Lemma 5.1 is a special case where $\ell = 1$ and $p = 1/2$. The above lemma is proved in a manner similar to that of Lemma 5.1. We provide a proof sketch in the remainder of the section.

We construct the row $\widehat{y}$ in $n$ iterations, finding a bit in each iteration. We start by setting $s_0 = \lambda$, where $\lambda$ is the empty string. Notice that the matrix $M$ has a row-side $\texttt{IC}\ y^*$ and that $y^*$

extends the empty string $s_0$. Thus, $\mu(s_0, \mathcal{Q}) = 1$. For $1 \leq k \leq n$, in iteration $k$, we find a prefix $s_k$ such that $\mu(s_k, \mathcal{Q}) \geq 1 - k\epsilon$, where $\epsilon$ is a parameter fixed below. By induction, assume that at the end of iteration $k-1$, we have found a prefix $s_{k-1}$ having goodness $\mu(s_{k-1}, \mathcal{Q}) \geq \rho$, where $\rho = 1-(k-1)\epsilon$. Observe that $\mu(s_{k-1}0, \mathcal{Q}) \geq \rho$ or $\mu(s_{k-1}1, \mathcal{Q}) \geq \rho$ (or both). In iteration $k$, we find a suitable bit $b \in \{0,1\}$ and append it to $s_{k-1}$ to get $s_k = s_{k-1}b$. The bit $b$ is found by invoking the algorithm given in Lemma 6.7: given a prefix $\beta$, the algorithm can distinguish between the cases of $\mu(\beta, \mathcal{Q}) \geq \rho$ and $\mu(\beta, \mathcal{Q}) \leq \rho - \epsilon$. We invoke the algorithm two times with $\beta = s_{k-1}0$ and $\beta = s_{k-1}1$ as inputs. By the above observation, the algorithm should answer "yes" on at least one of these invocations. Let $b$ be a bit such that the algorithm answers "yes" on input $\beta = s_{k-1}b$. We choose $b$ to be the required bit and set $s_k = s_{k-1}b$. It is easy to see that $\mu(s_k, \mathcal{Q}) \geq \rho - \epsilon$ (for otherwise, the algorithm should have output "no" on input $\beta = s_{k-1}b$).

At the end of the process we have a string $s_n$ such that $\mu(s_n, \mathcal{Q}) \geq 1 - n\epsilon$. Setting $\epsilon = (1-p)/n$, we see that $\mu(s_n, \mathcal{Q}) \geq p$. We output $\widehat{y} = s_n$.

## 6.3 Testing for the Goodness of a Prefix

In this section, we prove the following lemma used in the previous section. The lemma tests whether a prefix is good.

**Lemma 6.7** *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that solves the following problem. The input consists of: (i) a circuit succinctly representing a row-sided* $\mathrm{S}_2$*-type matrix* $M$ *of dimension* $2^n \times 2^m$*; (ii) a set* $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_\ell\}$*, where each* $Q_i \subseteq \{0,1\}^m$ *is a subset of the columns of* $M$ *that is presented succinctly in the form of a circuit; (iii) a string* $\beta$ *having length* $|\beta| \leq n$*; (iv) two parameters* $\rho \leq 1$ *and* $\epsilon > 0$*. It outputs an answer as follows:*

$$\begin{aligned} \mu(\beta, \mathcal{Q}) \geq \rho &\implies \textit{Answer="yes"} \\ \mu(\beta, \mathcal{Q}) \leq \rho - \epsilon &\implies \textit{Answer="no"}. \end{aligned}$$

*If* $\mu(\beta, \mathcal{Q})$ *falls in the interval* $[\rho - \epsilon, \rho]$*, the output can be arbitrary. The running time of the algorithm has a polynomial dependence on* $1/\rho$ *and* $1/\epsilon$*.*

The gap between the positive and negative instances in the above problem is only $\epsilon$. As in the case of Lemma 5.3, we amplify the gap via the running repeated trials and applying Chernoff bounds.

The amplification involves a parameter $r$, to be fixed shortly. Define a new matrix $\overline{M}$ as follows. Each row in $\overline{M}$ corresponds to a row in $M$ and each column of $\overline{M}$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_r \rangle$ of $r$ columns from $M$. Thus, $\overline{M}$ is of dimension $2^n \times 2^{\overline{m}}$, where $\overline{m} = mr$. For a row $y \in \{0,1\}^n$ and a column $\overline{z} = \langle z_1, z_2, \ldots z_r \rangle$, define the entry

$$\overline{M}[y, \overline{z}] \quad = \quad \begin{cases} 1 & \text{if } \frac{|\{z_i \,:\, y \text{ beats } z_i \text{ in } M\}|}{r} \geq (\rho - \frac{\epsilon}{2}) \\ 0 & \text{otherwise} \end{cases}$$

For a set of columns $X \subseteq \{0,1\}^m$, define $X^r$ to be the $r$-way Cartesian product of $X$:

$$X^r = \{\langle z_1, z_2, \ldots, z_r \rangle \;:\; \text{for } 1 \leq j \leq r,\, z_i \in X\}.$$

Notice that $X^r$ is a subset of columns of $\overline{M}$. For $1 \leq j \leq \ell$, define $\overline{Q}_j = (Q_j)^r$. Let $\overline{\mathcal{Q}} = \{\overline{Q}_1, \overline{Q}_2, \ldots, \overline{Q}_\ell\}$.

Set $r = \lceil 16m/\epsilon^2 \rceil$. Applying Chernoff bounds, we get the following claim, which expands the gap from $[\rho - \epsilon, \rho]$ to $[1/\overline{m}^4, 1/2]$.

**Lemma 6.8** *For any prefix $\beta$,*

$$\begin{aligned} \mu(\beta, \mathcal{Q}) \geq \rho &\implies \mu(\beta, \overline{\mathcal{Q}}) \geq 1/2 \\ \mu(\beta, \mathcal{Q}) \leq \rho - \epsilon &\implies \mu(\beta, \overline{\mathcal{Q}}) \leq 1/\overline{m}^4 \end{aligned}$$

The above lemma enables us to focus on $\overline{M}$ and $\overline{\mathcal{Q}}$, instead of $M$ and $\mathcal{Q}$. Our aim is to test whether the given prefix $\beta$ has an extension $y$ that beats a sufficiently large *fraction* of columns from each set $\overline{Q}_j$. This is accomplished in two successive steps. First, we estimate the size of each set $\overline{Q}_j$. In the second step, we test whether $\beta$ has an extension $y$ that beats a sufficiently large *number* of columns from each set $\overline{Q}_j$.

Our algorithm would make oracle queries to the following promise language. The promise language GoodPrefixGen is a generalization of the promise language GoodPrefix.

**Promise Language** GoodPrefixGen: The instances of this language consist of (i) a Boolean $2^n \times 2^m$ matrix $M$; (ii) a string $\beta$ having length $|\beta| \leq n$; (iii) a sequence of sets $X_1, X_2, \ldots, X_\ell$, where each set $X_i \subseteq \{0,1\}^m$ is presented succinctly in the form of a circuit; (iv) a sequence of positive integers $k_1, k_2, \ldots, k_\ell$.

*Positive instances:* There exists a row $y$ extending $\beta$ such that for all $1 \leq j \leq \ell$, $y$ beats at least $k_j 2^{k_j}$ columns in $X_j$.

*Negative instances:* For all rows $y$ extending $\beta$, there exists a set $X_j$ such that $y$ beats at most $2^{k_j - 1}$ columns in $X_j$.

The following lemma shows that GoodPrefixGen lies in the promise class prAM. The proof is similar to that of Lemma 5.6. We extend the proof of Lemma 5.6 to handle multiple subsets of columns in a parallel fashion.

**Lemma 6.9** *The promise language* GoodPrefix *belongs to the promise class* prAM.

*Proof:* We present an MAM protocol; it is well known such a protocol can be converted into an AM protocol [4].

For $1 \leq j \leq \ell$, let $\mathcal{H}_j$ be a family of universal hash functions from $\{0,1\}^m$ to $\{0,1\}^{k_j}$. Merlin sends a row $y \in \{0,1\}^n$ extending $\beta$. For $1 \leq j \leq \ell$, Arthur picks a set $H_j$ of hash functions uniformly and independently from $\mathcal{H}_j$. Then, Merlin must prove that $X_j$ has a collision under $H_j$, for each $1 \leq j \leq \ell$. If Merlin can accomplish this, then Arthur accepts the input; else Arthur rejects the input. Lemma 5.5 shows that: (i) if the input is a positive instance, then Arthur accepts with probability 1; (ii) if the input is a negative instance, then Arthur accepts with probability at most 1/2. □

We now describe the algorithm claimed in Lemma 6.7. For $1 \leq j \leq \ell$, we invoke the algorithm given in Theorem 3.2 on the set $\overline{Q}_j$ and find a estimate $U_j$ such that $(1 - \widehat{\delta})U_j \leq |\overline{Q}_j| \leq U_j$, where $\widehat{\delta}$ is a parameter suitably set as $\widehat{\delta} = 0.1$. For $1 \leq j \leq \ell$, fix $k_j = \lfloor \log \frac{(1-\widehat{\delta})U_j}{2\overline{m}} \rfloor$. The choice of $k_j$ ensures that for any row $y$

$$\begin{aligned} \mu(y, \overline{\mathcal{Q}}) \geq 1/2 &\implies \text{for all } 1 \leq j \leq \ell, y \text{ beats at least } k_j 2^{k_j} \text{ columns in } \overline{Q}_j \\ \mu(y, \overline{\mathcal{Q}}) \leq 1/\overline{m}^4 &\implies \text{there exists } 1 \leq j \leq \ell, y \text{ beats at most } 2^{k_j - 1} \text{ columns in } \overline{Q}_j \end{aligned}$$

We ask the GoodPrefix oracle a single query consisting of $\overline{M}$, $\beta$, $\overline{Q}_1, \overline{Q}_2, \ldots, \overline{Q}_\ell$ and $k_1, k_2, \ldots, k_\ell$. If the oracle answers "yes", we output "yes"; otherwise, we output "no". The correctness of the algorithm follows from Lemma 6.8.

# 7 Succinct Zero-sum Games

In this section, we present an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding near-optimal strategies for succinctly presented zero-sum games.

A two-player zero-sum 0-1 game is specified by a $2^n \times 2^m$ Boolean payoff matrix $M$. The rows and the columns correspond to the *pure strategies* of the row-player and the column-player, respectively. The row-player chooses a row $y \in \{0,1\}^n$ and the column-player chooses a column $z \in \{0,1\}^m$, simultaneously. The row-player then pays $M[y,z]$ to the column-player. The goal of the row-player is to minimize its loss, while the goal of the column-player is to maximize its gain.

A *mixed strategy* (or simply, a strategy) for the row-player is a probability distribution $P$ over the rows; similarly, a strategy for the column-player is a probability distribution $Q$ over the columns. The expected payoff is defined as:

$$M(P,Q) = \sum_{y,z} P(y)M[y,z]Q(z).$$

The classical Minmax theorem of von Neumann [23] says that even if the strategies are chosen sequentially, who plays first does not matter:

$$\min_P \max_Q M(P,Q) = \max_Q \min_P M(P,Q) = v^*,$$

where $v^*$ is called the *value* of the game. This means that there exist strategies $P^*$ and $Q^*$ such that

$$\max_Q M(P^*,Q) \le v^* \quad \text{and} \quad \min_P M(P,Q^*) \ge v^*.$$

Such strategies $P^*$ and $Q^*$ are called *optimal strategies*.

In some scenarios, it is sufficient to compute approximately optimal strategies. We shall be mainly concerned with additive errors.

**Definition 7.1** *A row-player strategy $\widetilde{P}$ is said to be $\epsilon$-optimal, if*

$$\max_Q M(\widetilde{P},Q) \le v^* + \epsilon,$$

*and similarly, a column-player strategy $\widetilde{Q}$ is said to be $\epsilon$-optimal, if*

$$\min_P M(P,\widetilde{Q}) \ge v^* - \epsilon,$$

We shall be interested in computing $\epsilon$-optimal strategies when the payoff matrix $M$ is presented *succinctly* in the form of a circuit $C$. Fortnow et. al [13] presented a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the above problem; their algorithm also finds an estimate of the value of the game within additive errors. Our main result is an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for the same problems. As mentioned in the introduction, our algorithm yields an alternative proof of the result by Fortnow et al. The details are also presented in this section.

The first task is to approximately find the value of the given game. This is treated next.

## 7.1 Approximately Finding the Value

We shall first discuss an $\mathrm{FP}^{\mathrm{prS}_2^p}$ algorithm for approximating the value. Here, $\mathrm{prS}_2^p$ refers to the promise version of the class $\mathrm{S}_2^p$.

**Definition 7.2** *A promise language $\Pi = (\Pi_1, \Pi_2)$ is said to be in the promise class $\mathrm{prS}_2^p$, 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*

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

*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. *(The complexity class $\mathrm{S}_2^p$ consists of languages in $\mathrm{prS}_2^p$.)*

Our $\mathrm{FP}^{\mathrm{prS}_2^p}$ algorithm for finding the value of a given game uses a promise language called SGV as the oracle. Fortnow et. al showed that this promise language lies in $\mathrm{prS}_2^p$, the promise version of the class $\mathrm{S}_2^p$.

**Succinct Game Value** (SGV) [6]: The input consists of a circuit succinctly representing a 0-1 zero-sum game and parameters $v$ and $\epsilon$.
*Positive instances:* $v^* \geq v + \epsilon$.
*Negative instances:* $v^* \leq v - \epsilon$.
Here, $v^*$ refers to the value of the given game.

**Theorem 7.3** *[13] The promise language SGV belongs to $\mathrm{prS}_2^p$.*

Using SGV as an oracle, we can perform a linear search in the interval $[0,1]$ and approximately find the value of a given game.

**Theorem 7.4** *There exists an $\mathrm{FP}^{\mathrm{prS}_2^p}$ algorithm that takes as input a circuit $C$ succinctly representing a 0-1 zero-sum game and a parameter $\epsilon$ and outputs a number $v$ such that $v - \epsilon \leq v^* \leq v + \epsilon$, where $v^*$ is the value of the given game. The running time of the algorithm is polynomial in $|C|$ and $1/\epsilon$.*

*Proof:* We shall use SGV as the $\mathrm{prS}_2^p$ oracle. Set $\epsilon' = \epsilon/2$. For $j = 0, 1, 2, \ldots, \lceil n/\epsilon' \rceil$, ask the query $\langle C, j\epsilon', \epsilon' \rangle$. Let $\widehat{j}$ be the first index such that the oracle answers "no". Set $v' = \widehat{j}\epsilon'$. Return$(v')$. Notice that $v'$ has the required property. □

Fortnow et. al [13] presented a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the problem of approximating the value of a succinct zero-sum game. We can show that $\mathrm{FP}^{\mathrm{prS}_2^p} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ (see Appendix). Thus, Theorem 7.4 provides a mild improvement over the previously best known result for the problem of approximating the value.

An easy extension of Theorem 4.9 shows that $\mathrm{FP}^{\mathrm{prS}_2^p} \subseteq \mathrm{FP}^{\mathrm{prAM}}$. Combined with Theorem 7.4, we get an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for approximating the value of succinct zero-sum games.

**Theorem 7.5** *There exists an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm that takes as input a circuit $C$ succinctly representing a 0-1 zero-sum game and a parameter $\epsilon$ and outputs a number $v$ such that $v - \epsilon \leq v^* \leq v + \epsilon$, where $v^*$ is the value of the given game. The running time of the algorithm is polynomial in $|C|$ and $1/\epsilon$.*

[6] $\epsilon$ is given as a rational number $a/b$, where $a$ and $b$ are specified in unary.

## 7.2 Finding Near-Optimal Strategies

In this section, we present an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding near-optimal strategies of a succinctly presented zero-sum game. The following small support lemma is useful in designing our algorithms.

A mixed strategy of a player is said to be *k-uniform*, if it chooses uniformly from a multi-set of $k$ pure strategies. Such a strategy can simply be specified by the multi-set of size $k$. The following lemma asserts the existence of $k$-uniform $\epsilon$-optimal, for a small value of $k$.

**Lemma 7.6 ([1][24][21])** *Let $M$ be a $0-1$ $2^n \times 2^m$ payoff matrix. Then there are $k$-uniform $\epsilon$-optimal strategies for both the row and the column-players, where $k = O(\frac{n+m}{\epsilon^2})$.*

The algorithm for finding near-optimal strategies goes via a reduction to the problem of finding a strong CIC of a given $\mathrm{S}_2$-type matrix. We then use the algorithm given in Theorem 6.3.

**Theorem 7.7** *There exists an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm that takes as input a circuit $C$ succinctly representing a $2^n \times 2^m$ payoff matrix $M$ of a 0-1 zero-sum game and a parameter $\epsilon$ and outputs a pair of $\epsilon$-optimal mixed strategies $(\widetilde{P}, \widetilde{Q})$. The running time of the algorithm is polynomial in $|C|$ and $1/\epsilon$.*

*Proof:* Let $v^*$ be the value of the game given by $M$. Our algorithm finds the required strategies $\widetilde{P}$ and $\widetilde{Q}$ in two phases. Here, we discuss the first phase of the algorithm that finds $\widetilde{P}$. The second phase for finding $\widetilde{Q}$ works in a similar manner.

The algorithm uses a parameter $\epsilon' = \epsilon/2$. Invoke the algorithm given in Theorem 7.5 with error parameter $\epsilon'/2$ and obtain an estimate $v$. Set $v^+ = v + \epsilon'/2$. Notice that $v^+$ is an upperbound on $v^*$ satisfying $v^* \leq v^+ \leq v^* + \epsilon'$. Invoking Lemma 7.6 with error parameter $\epsilon'$ we get a number $k = k(\epsilon')$ such that $M$ has $k$-uniform $\epsilon'$-optimal strategies $(P_{\epsilon'}, Q_{\epsilon'})$.

Construct a matrix $\overline{M}$ as follows. Each row $\overline{y}$ of $\overline{M}$ corresponds to a sequence $\langle y_1, y_2, \ldots, y_k \rangle$, where $y_i$ is a row in $M$; each column of $\overline{M}$ corresponds to a single column of $M$. Thus, $\overline{M}$ is a $2^{\overline{n}} \times 2^m$ matrix, where $\overline{n} = nk$. Its entries are defined as follows. Consider a row $\overline{y} = \langle y_1, y_2, \ldots, y_k \rangle$ and a column $z$. The entry $\overline{M}[\overline{y}, z]$ is defined as:

$$\overline{M}[\overline{y}, z] \quad = \quad \begin{cases} 1 & \text{if } \frac{1}{k}\left(\sum_{i=1}^{k} M[y_i, z]\right) \leq v^+ \\ 0 & \text{otherwise} \end{cases}$$

Let $\overline{y}$ be a row corresponding to $P_{\epsilon'}$. For any strategy $Q$ of the column-player the expected payoff $M(P_{\epsilon'}, Q) \leq v^+$. In particular, this is true for all pure strategies $z$ of the column-player. Therefore, we see that $\overline{y}$ is a row full of 1's. In other words, $\overline{M}$ is a row-side $\mathrm{S}_2$-type matrix.

Our next task is to find an $\alpha$-strong row-side CIC of the matrix $\overline{M}$, where $\alpha$ is a parameter suitably fixed as $\alpha = (1 - \epsilon/2)$. For this, we invoke the algorithm given in Theorem 6.3 on $\overline{M}$ and obtain a row-side $\alpha$-strong CIC $\overline{Y}$. Let the size of $\overline{Y}$ be $t$. Notice that each element $\overline{y}$ of $\overline{Y}$ is a sequence of $k$ pure row-player strategies. Consider the collection $S$ obtained by including all the pure strategies found in each $\overline{y} \in \overline{Y}$; thus, $S$ is a multiset of size $kt$. Let $\widetilde{P}$ be the $(kt)$-uniform strategy over the multiset $S$. We next prove that $\widetilde{P}$ is an $\epsilon$-optimal row-player strategy. The following easy claim is useful for this purpose.

**Claim 1:** Let $P$ be a $k$-uniform row-player strategy. Let $v \leq 1$ be such that for any pure column-player strategy $z$, $\Sigma_{y \in P} M[y, z] \leq v$. Then, $\max_Q M(P, Q) \leq v$, where $Q$ ranges over all mixed strategies of the column-player.

**Claim 2:** $\widetilde{P}$ is an $\epsilon$-optimal row-player strategy. *Proof:* Consider any pure strategy of the column-player $z \in \{0,1\}^m$. Since $\overline{Y}$ is an $\alpha$-strong CIC, at least an $\alpha$ fraction of the rows in $\overline{Y}$ beat $z$. A row $\overline{y} \in \overline{Y}$ beating $z$ means that $\overline{y} = \{y_1, y_2, \ldots, y_k\}$ satisfies $\sum_{y_i \in \overline{y}} M[y_i, z] \leq kv^+$. Recall that $v^+ \leq v^* + \epsilon'$. We now want to estimate the sum $\sum_{y \in S} M[y, z]$, which can be written as:

$$\sum_{y \in S} M[y,z] = \sum_{\overline{y} \in \overline{Y}} \sum_{y_i \in \overline{y}} M[y_i, z].$$

To estimate the sum on the RHS, we partition $\overline{Y}$ into two disjoint sets $\overline{Y}_{good}$ and $\overline{Y}_{bad}$: place all the $\overline{y} \in \overline{Y}$ that beat $z$ in $\overline{Y}_{good}$ and the rest in $\overline{Y}_{bad}$. Notice that $|\overline{Y}_{good}| \geq \alpha t$.

$$\begin{aligned}
\sum_{y \in S} M[y,z] &= \sum_{\overline{y} \in \overline{Y}} \sum_{y_i \in \overline{y}} M[y_i, z] \\
&= \sum_{\overline{y} \in \overline{Y}_{good}} \sum_{y_i \in \overline{y}} M[y_i, z] + \sum_{\overline{y} \in \overline{Y}_{bad}} \sum_{y_i \in \overline{y}} M[y_i, z] \\
&\leq \sum_{\overline{y} \in \overline{Y}_{good}} kv^+ + \sum_{\overline{y} \in \overline{Y}_{bad}} k \\
&\leq |\overline{Y}_{good}| kv^+ + |\overline{Y}_{bad}| k \\
&\leq tk(v^* + \epsilon') + (1-\alpha)tk \leq tk(v^* + \epsilon)
\end{aligned}$$

The last inequality follows from the choice of $\alpha$ and $\epsilon'$. Now, Claim 2 follows from Claim 1.

The second phase of the algorithm that finds $\widetilde{Q}$ works in a similar manner. Fix $\epsilon' = \epsilon/2$. Using the algorithm given in Theorem 7.5, we get a good lowerbound $v^-$ on $v^*$ satisfying $v^* - \epsilon' \leq v^- \leq v^*$. Invoking Lemma 7.6, we get a number $k = k(\epsilon')$ such that $M$ has $k$-uniform $\epsilon'$-optimal strategies. Define a $2^n \times 2^{\overline{m}}$ matrix $\overline{M}$, where $\overline{m} = mk$. For a row $y$ and a column $\overline{z} = \langle z_1, z_2, \ldots, z_k \rangle$, the entry $\overline{M}[y, \overline{z}]$ is defined as:

$$\overline{M}[y,\overline{z}] = \begin{cases} 0 & \text{if } \frac{1}{k}\left(\sum_{z_i} M[y, z_i]\right) \geq v^- \\ 1 & \text{otherwise} \end{cases}$$

Observe that the matrix $\overline{M}$ is a column-side $S_2$-type matrix. Our next task is to compute a column-side $\alpha$-strong CIC $\overline{Z}$ of the matrix $\overline{M}$, where $\alpha$ is suitably fixed as $\alpha = 1 - \epsilon/2$. For this, we again appeal to Theorem 6.3. Let the size of $\overline{Z}$ be $t$. Consider the collection $S$ obtained by including all the pure strategies found in each $\overline{z} \in \overline{Z}$. Let $\widetilde{Q}$ be the $(kt)$-uniform strategy over the multiset $S$. We next show that $\widetilde{Q}$ is an $\epsilon$-optimal column-player strategy. The following claim is useful for this purpose.

**Claim 3:** Let $Q$ be a $k$-uniform row-player strategy. Let $v \leq 1$ be such that for any pure row-player strategy $y$, $\Sigma_{z \in Q} M[y, z] \leq v$. Then, $\max_P M(P, Q) \leq v$, where $P$ ranges over all mixed strategies of the row-player.

**Claim 4:** $\widetilde{Q}$ is an $\epsilon$-optimal column-player strategy. *Proof:* Consider any pure strategy of the row-player $y \in \{0,1\}^n$. We partition $\overline{Z}$ into two disjoint sets $\overline{Z}_{good}$ and $\overline{Z}_{bad}$: denote by $\overline{Z}_{good}$ the set of all $\overline{z} \in \overline{Z}$ that beat $y$ and $\overline{Z}_{bad} = \overline{Z} - \overline{Z}_{good}$. Notice that $|\overline{Z}_{good}| \geq \alpha t$. A column $\overline{z} \in \overline{Z}$ beating $y$ means that $\overline{z} = \{z_1, z_2, \ldots, z_k\}$ satisfies $\sum_{z_i \in \overline{z}} M[y, z_i] \geq kv^-$. We have

$$\begin{aligned}
\sum_{z \in S} M[y,z] &= \sum_{\overline{z} \in \overline{Z}} \sum_{z_i \in \overline{z}} M[y,z_i] \\
&= \sum_{\overline{z} \in \overline{Z}_{good}} \sum_{z_i \in \overline{z}} M[y,z_i] + \sum_{\overline{z} \in \overline{Z}_{bad}} \sum_{z_i \in \overline{z}} M[y,z_i] \\
&\geq \sum_{\overline{z} \in \overline{Z}_{good}} kv^- + \sum_{\overline{z} \in \overline{Z}_{bad}} 0 \\
&\geq |\overline{Z}_{good}| kv^- \\
&\geq \alpha tk(v^* - \epsilon') \\
&\geq tk(v^* - \epsilon)
\end{aligned}$$

The last inequality follows from the choice of $\alpha$ and $\epsilon'$, and the fact that $v^* \leq 1$. Now Claim 4 follows from Claim 3. □

### 7.3 An Alternative $\mathrm{ZPP}^{\mathrm{NP}}$ Algorithm for Finding Near-Optimal Strategies

Fortnow et. al [13] designed a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the problem of finding near-optimal strategies considered in Theorem 7.7. By suitably adapting the proof of Theorem 7.7, we derive an alternative proof of this result. To achieve this, we make use of Cai's $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm [6] for finding strong `CIC` (Theorem 6.2). Next we sketch the alternative proof.

**Theorem 7.8** *[13] There exists an $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm that takes as input a circuit $C$ succinctly representing a $2^n \times 2^m$ payoff matrix $M$ of a 0-1 zero-sum game and a parameter $\epsilon$ and outputs a pair of $\epsilon$-optimal mixed strategies $(\widetilde{P}, \widetilde{Q})$. The running time of the algorithm is polynomial in $|C|$ and $1/\epsilon$.*

*Proof:* We adapt the proof of Theorem 7.7. The first step is to approximately find the value of a given game; namely, we need a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the problem considered in Theorem 7.5. Theorem 7.4 presents a $\mathrm{FP}^{\mathrm{prS}_2^p}$ for this problem. By extending Cai's result [6] that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}$, we can show that $\mathrm{FP}^{\mathrm{prS}_2^p}$ can be simulated in $\mathrm{ZPP}^{\mathrm{NP}}$ (see Appendix). Thus, we get a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for approximating the value of a given game.

To find the near-optimal strategies, we follow the algorithm given in Theorem 7.7. Notice that the above algorithm runs in polynomial time, given a black-box for computing $\alpha$-strong `CIC`'s for $\mathrm{S}_2$-type matrices. In the original proof, we made use of the $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding strong `CIC`s given by Theorem 6.3. Instead, here we use Cai's $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm (Theorem 6.2) for finding the required strong `CIC`s. This yields a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for finding the $\epsilon$-optimal strategies $\widetilde{P}$ and $\widetilde{Q}$. □

## 8 Conditional Derandomization

As mentioned in the introduction, our main results can be derandomized under suitable hardness hypotheses. For the ease of exposition, we had presented slightly weaker derandomization results in the introduction. Here, we state the actual stronger results. These are obtained by combining our results with the hitting set generator construction of Miltersen and Vinodchandran [22]. The following result directly follows from their construction.

**Theorem 8.1 ([22])** • *Suppose there exists a language $L$ computable in $\mathrm{E}_{||}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, $\mathrm{P}_{||}^{\mathrm{prAM}} = \mathrm{P}_{||}^{\mathrm{NP}}$.*

- *Suppose there exists a language $L$ computable in $\mathrm{E}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, $\mathrm{P}^{\mathrm{prAM}} = \mathrm{P}^{\mathrm{NP}}$.*

- *Suppose there exists a language $L$ computable in $\mathrm{E}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, $\mathrm{FP}^{\mathrm{prAM}} = \mathrm{FP}^{\mathrm{NP}}$.*

The following surprising result was first proved by Shaltiel and Umans [29]. We obtain an alternative proof by combining Theorem 8.1 with Theorem 3.4 and 4.9.

**Theorem 8.2 ([29])** • *Suppose there exists a language $L$ computable in $\mathrm{E}_{||}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, $\mathrm{BPP}_{||}^{\mathrm{NP}} = \mathrm{P}_{||}^{\mathrm{NP}}$.*

- *Suppose there exists a language $L$ computable in $\mathrm{E}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, $\mathrm{S}_2^p = \mathrm{P}^{\mathrm{NP}}$.*

The following result offers a conditional derandomization of Theorem 7.7. It shows how to find $\epsilon$-optimal strategies of a succinctly presented zero-sum game in $\mathrm{FP}^{\mathrm{NP}}$, under a suitable hardness hypothesis. It is not clear whether the following derandomization can be achieved for the $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm by Fortnow et al. [13] (given in Theorem 7.8).

**Theorem 8.3** *Suppose there exists a language $L$ computable in $\mathrm{E}^{\mathrm{NP}}$ and an $\epsilon > 0$ such that for sufficiently large $n$, SV-nondeterministic circuits of size $2^{\epsilon n}$ cannot compute $L \cap \{0,1\}^n$. Then, there exists an $\mathrm{FP}^{\mathrm{NP}}$ algorithm that takes as input a circuit $C$ succinctly representing a $2^n \times 2^m$ payoff matrix $M$ of a 0-1 zero-sum game and a parameter $\epsilon$ and outputs a pair of $\epsilon$-optimal mixed strategies $(\widetilde{P}, \widetilde{Q})$. The running time of the algorithm is polynomial in $|C|$ and $1/\epsilon$.*

# 9 Open Problems

Here we state some open problems. We showed that $\mathrm{BPP}_{||}^{\mathrm{NP}} \subseteq \mathrm{P}_{||}^{\mathrm{prAM}}$. The most interesting and challenging open problem asks whether $\mathrm{BPP}^{\mathrm{NP}}$ is contained in $\mathrm{P}^{\mathrm{prAM}}$. An affirmative answer would have two interesting implications. First, this would show that $\mathrm{BPP}^{\mathrm{NP}} \subseteq \mathrm{P}^{\Sigma_2^p}$. The second implication is that $\mathrm{BPP}^{\mathrm{NP}}$ can be derandomized under the hardness hypothesis used for derandomizing AM.

We presented a $\mathrm{FP}^{\mathrm{prS}_2^p}$ algorithm for approximating the value of a succinct zero-sum game. It is open whether near-optimal strategies can be found in $\mathrm{FP}^{\mathrm{prS}_2^p}$. We know that $\mathrm{P}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{P}^{\mathrm{prS}_2^p} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$, and also that $\mathrm{P}^{\mathrm{MA}} \subseteq \mathrm{S}_2^p$ and $\mathrm{P}^{\mathrm{S}_2^p} \subseteq \mathrm{S}_2^p$. It is open whether $\mathrm{P}^{\mathrm{prMA}}$ and $\mathrm{P}^{\mathrm{prS}_2^p}$ are contained in $\mathrm{S}_2^p$.

## Acknowledgments

We thank Eric Allender, Dieter van Melkebeek and N.V.Vinodchandran for useful comments and suggestions. We also thank Dieter van Melkebeek for sharing with us his alternative proof of Theorem 3.4.

## References

[1] I. Althöfer. On sparse approximations to randomized strategies and convex combinations. *Linear Algebra and its Applications*, 199, 1994.

[2] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. *Theoretical Computer Science*, 255(1-2):205–221, 2001.

[3] 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.

[4] 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.

[5] 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.

[6] J. Cai. $S_2^p \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 73(1), 2007.

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

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

[9] V. Chakaravarthy and S. Roy. Arthur and merlin as oracles. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science*, 2008.

[10] V. Chakaravarthy and S. Roy. Finding irrefutable certificates for $S_2^p$ via Arthur and Merlin. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, 2008.

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

[12] J. Feigenbaum, D. Koller, and P. Shor. A game-theoretic classification of interactive complexity classes. In *Proceedings of the 10th Annual IEEE Conference on Computational Complexity*, 1995.

[13] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans. On the complexity of succinct zero-sum games. *Computational Complexity*, 17(3):353–376, 2008.

[14] L. Fortnow, A. Pavan, and S. Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. *Journal of Computer and System Sciences*, 74(3):358–363, 2008.

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

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

[17] R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.

[18] 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.

[19] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

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

[21] R. Lipton and N. Young. Simple strategies for large zero-sum games with applications to complexity theory. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, 1994.

[22] P. Miltersen and N. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[23] J. Neumann. Zur theorie der gesellschaftspiel. *Mathematische Annalen*, 100, 1928.

[24] J. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39:67–71, 1991.

[25] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[26] G. Owen. *Game Theory*. Academic Press, 1982.

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

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

[29] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4):298–341, 2007.

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

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

# A Upperbounds for $\mathrm{P}^{\mathrm{prAM}}$ and related classes

Here, we catalogue some upperbounds for $\mathrm{P}^{\mathrm{prAM}}$ and related classes. We start with the following simple result.

**Theorem A.1**

- $\mathrm{P}^{\mathrm{prAM}} \subseteq \mathrm{BPP}^{\mathrm{NP}}$.
- $\mathrm{P}_{||}^{\mathrm{prAM}} \subseteq \mathrm{BPP}_{||}^{\mathrm{NP}}$.
- $\mathrm{FP}^{\mathrm{prAM}}$ *is contained in the function class* $\mathrm{BPP}^{\mathrm{NP}}$.
- $\mathrm{FP}_{||}^{\mathrm{prAM}}$ *is contained in the function class* $\mathrm{BPP}_{||}^{\mathrm{NP}}$.

Next, we turn our attention to $\mathrm{P}^{\mathrm{prMA}}$. Let us first recall some known results: (i) $\mathrm{MA} \subseteq \mathrm{S}_2^p$ [28]; (ii) $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ [6]; (iii) $\mathrm{MA} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ [16, 2]. An easy extension of (i) shows that $\mathrm{prMA} \subseteq \mathrm{prS}_2^p$. Below, we extend (ii) as $\mathrm{P}^{\mathrm{prS}_2^p} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. Thus, we get a generalization of (iii) giving $\mathrm{P}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

**Theorem A.2** $\mathrm{P}^{\mathrm{prS}_2^p} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

*Proof:* We sketch the proof here. It is straightforward to show that the following promise language is $\mathrm{prS}_2^p$-hard.

**Promise language** $\mathrm{S}_2^p$-MembershipTest:The input consists of circuit succinctly representing a Boolean matrix $M$.
*Positive instance:* $M$ is a row-side $\mathrm{S}_2$-type matrix.
*Negative instance:* $M$ is a column-side $\mathrm{S}_2$-type matrix.

Cai's result [6] that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ provides a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for the above problem. The algorithm has the following characteristics. (i) If the input matrix $M$ is a row-side $\mathrm{S}_2$-type matrix, then the output is either "row-side" or "?", where "?" will be output with small probability. (ii) If the input matrix $M$ is a column-side $\mathrm{S}_2$-type matrix, then the output is either "column-side" or "?", where "?" will be output with small probability. (iii) If the input matrix is not an $\mathrm{S}_2$-type matrix, then the output can be "row-side", "column-side" or "?", with no guarantees on the probability with which these possibilities will be output (in particular, "?" may be output with high probability).

Consider a language $L \in \mathrm{P}^{\mathrm{prS}_2^p}$. Let $\mathcal{A}$ be a polynomial time algorithm which given oracle access to $\mathrm{S}_2^p$-MembershipTest computes $L$. We shall design a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for $L$. Let $C$ be a query raised by $\mathcal{A}$, where $C$ succinctly represents a Boolean matrix $M$ of size $2^n \times 2^m$. Clearly, we can simulate Cai's $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm on $C$. A concern here is that $M$ may not be a $\mathrm{S}_2$-type matrix and so, Cai's algorithm may output an arbitrary answer as "row-side" or "column-side". This is not an issue, since the algorithm $\mathcal{A}$ should be able to tolerate either answer, by definition. The more serious issue is that Cai's algorithm may output "?" with high probability.

We handle this issue by appealing to Lemma 4.14, due to Goldreich and Wigderson [15]. Using the above lemma, we convert the given matrix into two matrices at least one of which is guaranteed to be an $\mathrm{S}_2$-type matrix. The construction is similar to the one used in the proof of Theorem 4.15.

We define two Boolean matrices $\overline{M}_1$ and $\overline{M}_2$ as follows. The matrix $\overline{M}_1$ is of size $2^{nm} \times 2^m$. Each row of $\overline{M}_1$ corresponds to a sequence $\langle y_1, y_2, \ldots, y_m \rangle$ of $m$ rows of $M$. Each column $z$ of

$\overline{M}_1$ corresponds to a single column of $M$. The entries of $\overline{M}_1$ are defined as below. For a row $\langle y_1, y_2, \ldots, y_m \rangle \in \{0,1\}^{nm}$ and a column $z \in \{0,1\}^m$, the entry is defined as:

$$\overline{M}_1[\langle y_1, y_2, \ldots, y_m \rangle, z] \quad = \quad \begin{cases} 1 & \text{if some } y_i \text{ beats } z \text{ in } M \\ 0 & \text{otherwise} \end{cases}$$

The matrix $\overline{M}_2$ is defined analogously. Each row $y$ of $\overline{M}_1$ corresponds to a single row of $M$. Each column of $\overline{M}_2$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_n \rangle$ of $n$ columns of $M$. Thus, $\overline{M}_2$ is matrix of size $2^n \times 2^{mn}$. The entries of $\overline{M}_2$ are defined as below. For a row $y$ and a column $\langle z_1, z_2, \ldots, z_n \rangle \in \{0,1\}^{mn}$, the entry is defined as:

$$\overline{M}_2[y, \langle z_1, z_2, \ldots, z_n \rangle] \quad = \quad \begin{cases} 0 & \text{if some } z_i \text{ beats } y \text{ in } M \\ 1 & \text{otherwise} \end{cases}$$

Observe that the following claims are true. (i) If $M$ is a row-side $S_2$-type matrix then both $\overline{M}_1$ and $\overline{M}_2$ are row-side $S_2$-type matrices; (ii) If $M$ is a column-side $S_2$-type matrix then both $\overline{M}_1$ and $\overline{M}_2$ are column-side $S_2$-type matrices; (iii) If $M$ is not an $S_2$-type matrix then $\overline{M}_1$ is a row-side $S_2$-type matrix or $\overline{M}_2$ is a column-side $S_2$-type matrix (or both) - this follows from Theorem 4.14.

We can construct in polynomial time circuits $\overline{C}_1$ and $\overline{C}_2$ that succinctly encode the matrices $\overline{M}_1$ and $\overline{M}_2$, respectively. We run Cai's algorithm on both these circuits. Let $s_1$ and $s_2$ be the outcome of these two runs. Apply the following procedure to find an "unified" outcome $s$:

- *Case 1:* At least one of $s_1$ or $s_2$ is "row-side". In this case, set $s$="row-side".
- *Case 2:* Neither $s_1$ and $s_2$ is "row-side", but at least one of $s_1$ or $s_2$ is "column-side". In this case, set $s$="column-side".
- *Case 3:* Both $s_1$ and $s_2$ are "?". In this case, set $s$="?".

If $s$="?", then output "?" and stop simulating $\mathcal{A}$. Otherwise, take $s$ to be the answer to the query $C$ and continue simulating $\mathcal{A}$.

It can now be argued that our algorithm is a $\mathrm{ZPP}^{\mathrm{NP}}$ algorithm for $L$. In particular, on any input, it would output "?" with only a low probability. □

**Corollary A.3** $\mathrm{P}^{\mathrm{prMA}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$.

The above theorems can be extended for function classes.

**Theorem A.4**
- $\mathrm{FP}^{\mathrm{prS}_2^p}$ *is contained in the function class* $\mathrm{ZPP}^{\mathrm{NP}}$.
- $\mathrm{FP}^{\mathrm{prMA}}$ *is contained in the function class* $\mathrm{ZPP}^{\mathrm{NP}}$.

# B Proving SAT Does not Have Small Size Circuits

In [14], Fortnow, Pavan and Sengupta showed that if $\mathrm{P}^{\mathrm{NP}[1]} = \mathrm{P}^{\mathrm{NP}[2]}$ then $\mathrm{PH} = \mathrm{S}_2^p$. This was shown by building on a key lemma that provides a mechanism for proving that SAT does not have small size circuits, if that is the case. Here we derive an alternative proof of this result as a corollary of Theorem 4.14, due to Goldreich and Wigderson [15].

Let $C$ be a (possibly incorrect) circuit claimed to compute SAT at certain length $n$. Recall that $C$ is *nice*, if $C$ does not accept unsatisfiable formulas.

**Lemma B.1** *[14] Fix $n > 0$. For every $k > 0$, if* SAT *does not have $n^{k+2}$ size circuits at length $n$, then there exists a set $S$ of satisfiable formulas of length $n$, called counter-examples, such that every nice circuit of size $n^k$ is wrong on at least one formula from $S$. The cardinality of $S$ is polynomial in $n$.*

*Proof:* Fix a length $n$. Construct a Boolean matrix $M$ as follows. Each satisfiable formula of length $n$ constitutes a row in $M$. Each nice circuit of size at most $n^k$ constitutes a column in $M$. Thus $M$ has at most $2^n$ rows and at most $2^{n^k}$ columns. For a satisfiable formula $\varphi$ and a nice circuit $C$, the entry $M[\varphi, C]$ is defined as:

$$M[\varphi, C] \quad = \quad \begin{cases} 1 & \text{if } C(\varphi) = \textit{reject} \\ 0 & \text{if } C(\varphi) = \textit{accept} \end{cases}$$

By Theorem 4.14, the matrix $M$ has at least one of the following: (i) a row-side `CIC` of size at most $n^k$; (ii) a column-side `CIC` of size at most $n$. Suppose (ii) happens. Let $\mathcal{C}$ be the assumed column-side `CIC` of size $n$. Construct a circuit $C^*$ that works as follows: given a formula $\varphi$ of length $n$, $C^*$ accepts $\varphi$, if at least one circuit in $\mathcal{C}$ accepts it; otherwise $\varphi$ is rejected. In other words, $C^*$ is the OR of all the circuits in $\mathcal{C}$. Notice that $C^*$ is a correct circuit for SAT and that the size of $C^*$ is $n^{k+1}$. This contradicts the assumption that SAT at length $n$ does not have circuits of size $n^{k+2}$. Thus, the only possibility is (i): there exists a row-side `CIC` $S$ of cardinality at most $n^k$. Observe that every nice circuit is incorrect on at least one formula in $S$. We take $S$ to be the claimed set of counter-examples. □