A study of Merlin-Arthur Protocols for K-SAT

Aritra Majumder
Supervised by: Prof. V. Arvind

Chennai Mathematical Institute

July 3, 2025

Strong Exponential Time Hypothesis and Extensions

- Given a SAT instance, certifying its satisfiability (SAT) is in NP.

Strong Exponential Time Hypothesis and Extensions

- Given a SAT instance, certifying its satisfiability (SAT) is in NP.
- We can just give a truth assignment as a certificate.

Strong Exponential Time Hypothesis and Extensions

- Given a SAT instance, certifying its satisfiability (SAT) is in NP.
- We can just give a truth assignment as a certificate.
- However, certifying that it is unsatisfiable (UNSAT) is in co-NP and, therefore, is likely to be a harder problem.

Strong Exponential Time Hypothesis and Extensions

- Given a SAT instance, certifying its satisfiability (SAT) is in NP.
- We can just give a truth assignment as a certificate.
- However, certifying that it is unsatisfiable (UNSAT) is in co-NP and, therefore, is likely to be a harder problem.
- Carmosino et al. proposed the Non-deterministic Strong Exponential Hypothesis, **NSETH**, which says that there are no proof systems that can refute unsatisfiable k-SAT instances significantly more efficiently than the enumeration of all variable assignments.

Strong Exponential Time Hypothesis and Extensions

- Given a SAT instance, certifying its satisfiability (SAT) is in NP.
- We can just give a truth assignment as a certificate.
- However, certifying that it is unsatisfiable (UNSAT) is in co-NP and, therefore, is likely to be a harder problem.
- Carmosino et al. proposed the Non-deterministic Strong Exponential Hypothesis, **NSETH**, which says that there are no proof systems that can refute unsatisfiable k-SAT instances significantly more efficiently than the enumeration of all variable assignments.
- Carmosino also proposed **MASETH** and **AMSETH** via private communication to Williams, which says that there are no AM and MA protocols that can refute satisfiability of unsatisfiable k-SAT instances significantly more efficiently than the enumeration of all variable assignments.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.
- Verifier then checks if two circuits agree on a random point using a small amount of randomness, with low chance of error.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.
- Verifier then checks if two circuits agree on a random point using a small amount of randomness, with low chance of error.
- This proof system leads to a significant finding: *MASETH is False*.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.
- Verifier then checks if two circuits agree on a random point using a small amount of randomness, with low chance of error.
- This proof system leads to a significant finding: *MASETH is False*.
- This means for Boolean circuits there's an efficient way to prove the number of satisfying assignments.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.
- Verifier then checks if two circuits agree on a random point using a small amount of randomness, with low chance of error.
- This proof system leads to a significant finding: *MASETH is False*.
- This means for Boolean circuits there's an efficient way to prove the number of satisfying assignments.
- Specifically, a Prover can provide a proof (of size about $2^{n/2}$) for the claimed count of SAT assignments.

William's Result

- Efficiently convince a Verifier about the evaluations of an arithmetic circuit C at multiple input points $(a_1, \ldots, a_k)$.
- The Prover sends the claimed circuit outputs $C(a_1), \ldots, C(a_k)$ and a relatively short proof which is a univariate polynomial.
- Verifier then checks if two circuits agree on a random point using a small amount of randomness, with low chance of error.
- This proof system leads to a significant finding: *MASETH is False*.
- This means for Boolean circuits there's an efficient way to prove the number of satisfying assignments.
- Specifically, a Prover can provide a proof (of size about $2^{n/2}$) for the claimed count of SAT assignments.
- A Verifier can check in about $2^{n/2}$ time using a small number of random bits ($O(n)$), with a very small error probability.

Brief Overview

1. First, we will go through William's MA protocol for SAT.

Brief Overview

1. First, we will go through William's MA protocol for SAT.
2. Then, we will review Akmal et. al.'s work on improving the protocol.

Brief Overview

1. First, we will go through William's MA protocol for SAT.
2. Then, we will review Akmal et. al.'s work on improving the protocol.
3. Next, we will go through an MA protocol for SUB-SAT problem.

Brief Overview

1. First, we will go through William's MA protocol for SAT.
2. Then, we will review Akmal et. al.'s work on improving the protocol.
3. Next, we will go through an MA protocol for SUB-SAT problem.
4. Next, we will go through an MA protocol for POLY-EQS problem.

MASETH is False

Batch Evaluation Protocol

Preliminaries

Let us first present two algorithmic results.

Preliminaries

Let us first present two algorithmic results.

Fast Multipoint Evaluation of Univariate Polynomials

Given a polynomial $p(x) \in F[X]$ with $\deg(p) \leq n$, presented as a vector of coefficients $[a_0, \ldots, a_{\deg(p)}]$, and given points $\alpha_1, \ldots, \alpha_n \in F$, we can output the vector $(p(\alpha_1), \ldots, p(\alpha_n)) \in F^n$ in $O(\mathrm{mult}(n) \cdot \log n)$ additions, multiplications in F. This algorithm was developed by Borodin & Moenck.

Preliminaries

Let us first present two algorithmic results.

Fast Multipoint Evaluation of Univariate Polynomials

Given a polynomial $p(x) \in F[X]$ with $\deg(p) \leq n$, presented as a vector of coefficients $[a_0, \ldots, a_{\deg(p)}]$, and given points $\alpha_1, \ldots, \alpha_n \in F$, we can output the vector $(p(\alpha_1), \ldots, p(\alpha_n)) \in F^n$ in $O(\mathrm{mult}(n) \cdot \log n)$ additions, multiplications in F. This algorithm was developed by Borodin & Moenck.

Fast Univariate Interpolation

Given a set of pairs $\{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$ with all α_i distinct, we can output the coefficients of $p(x) \in F[X]$ of degree at most n satisfying $p(\alpha_i) = \beta_i$ for all i, in $O(\mathrm{mult}(n) \cdot \log^2 n)$ additions and multiplications in F. This algorithm was developed by Horowitz.

Batch Evaluation Protocol

Batch Evaluation

For every prime power q and $\varepsilon > 0$, Multipoint Circuit Evaluation for K points in F_q^n on an arithmetic circuit C of n inputs, s gates, and degree d has an MA-proof system where:

- Merlin sends a proof of $O(Kd \cdot \log(Kqd/\varepsilon))$ bits, and
- Arthur tosses at most $\log(Kqd/\varepsilon)$ coins, outputs $(C(\alpha_1), \ldots, C(\alpha_K))$ incorrectly with probability at most ε, and runs in time $K \cdot \max\{d, n\} + s \cdot \mathrm{poly}(\log s) \cdot \mathrm{poly}(\log(Kqd/\varepsilon))$.

Batch Evaluation Protocol

Batch Evaluation

For every prime power q and $\varepsilon > 0$, Multipoint Circuit Evaluation for K points in F_q^n on an arithmetic circuit C of n inputs, s gates, and degree d has an MA-proof system where:

- Merlin sends a proof of $O(Kd \cdot \log(Kqd/\varepsilon))$ bits, and
- Arthur tosses at most $\log(Kqd/\varepsilon)$ coins, outputs $(C(\alpha_1), \ldots, C(\alpha_K))$ incorrectly with probability at most ε, and runs in time $K \cdot \max\{d, n\} + s \cdot \mathrm{poly}(\log s) \cdot \mathrm{poly}(\log(Kqd/\varepsilon))$.

I will first go through the proof sketch.

Batch Evaluation Protocol

Batch Evaluation

For every prime power q and $\varepsilon > 0$, Multipoint Circuit Evaluation for K points in F_q^n on an arithmetic circuit C of n inputs, s gates, and degree d has an MA-proof system where:

- Merlin sends a proof of $O(Kd \cdot \log(Kqd/\varepsilon))$ bits, and
- Arthur tosses at most $\log(Kqd/\varepsilon)$ coins, outputs $(C(\alpha_1), \ldots, C(\alpha_K))$ incorrectly with probability at most ε, and runs in time $K \cdot \max\{d, n\} + s \cdot \mathrm{poly}(\log s) \cdot \mathrm{poly}(\log(Kqd/\varepsilon))$.

I will first go through the proof sketch.

- Given a multivariate polynomial $C(x_1, \ldots, x_n)$ over a field F and K input points $a_1, \ldots, a_K \in F^n$, we want to evaluate C on all these points efficiently using a Merlin-Arthur (MA) proof system.

Batch Evaluation Protocol

Batch Evaluation

For every prime power q and $\varepsilon > 0$, Multipoint Circuit Evaluation for K points in F_q^n on an arithmetic circuit C of n inputs, s gates, and degree d has an MA-proof system where:

- Merlin sends a proof of $O(Kd \cdot \log(Kqd/\varepsilon))$ bits, and
- Arthur tosses at most $\log(Kqd/\varepsilon)$ coins, outputs $(C(\alpha_1), \ldots, C(\alpha_K))$ incorrectly with probability at most ε, and runs in time $K \cdot \max\{d, n\} + s \cdot \text{poly}(\log s) \cdot \text{poly}(\log(Kqd/\varepsilon))$.

I will first go through the proof sketch.

- Given a multivariate polynomial $C(x_1, \ldots, x_n)$ over a field F and K input points $a_1, \ldots, a_K \in F^n$, we want to evaluate C on all these points efficiently using a Merlin-Arthur (MA) proof system.
- The verifier defines a canonical mapping between each $a_i \in F^n$ and a unique $\alpha_i \in S \subseteq F$, where $|S| = K$.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)
- Define $R(x) := C(\Psi_1(x), \ldots, \Psi_n(x)) \in F[x]$, a univariate polynomial satisfying $R(\alpha_i) = C(a_i)$ for all i.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)
- Define $R(x) := C(\Psi_1(x), \dots, \Psi_n(x)) \in F[x]$, a univariate polynomial satisfying $R(\alpha_i) = C(a_i)$ for all i.
- Merlin sends a polynomial $Q(x)$ of degree at most dK claiming $Q(x) = R(x)$. Arthur randomly picks $r \in F$ and checks if $Q(r) = C(\Psi_1(r), \dots, \Psi_n(r))$.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)
- Define $R(x) := C(\Psi_1(x), \ldots, \Psi_n(x)) \in F[x]$, a univariate polynomial satisfying $R(\alpha_i) = C(a_i)$ for all i.
- Merlin sends a polynomial $Q(x)$ of degree at most dK claiming $Q(x) = R(x)$. Arthur randomly picks $r \in F$ and checks if $Q(r) = C(\Psi_1(r), \ldots, \Psi_n(r))$.
- Arthur evaluates $\Psi_j(r)$ for all j using Horner's rule, and then evaluates C on these values by simulating the arithmetic circuit over F.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)
- Define $R(x) := C(\Psi_1(x), \ldots, \Psi_n(x)) \in F[x]$, a univariate polynomial satisfying $R(\alpha_i) = C(a_i)$ for all i.
- Merlin sends a polynomial $Q(x)$ of degree at most dK claiming $Q(x) = R(x)$. Arthur randomly picks $r \in F$ and checks if $Q(r) = C(\Psi_1(r), \ldots, \Psi_n(r))$.
- Arthur evaluates $\Psi_j(r)$ for all j using Horner's rule, and then evaluates C on these values by simulating the arithmetic circuit over F.
- If $Q(r) \neq R(r)$, Arthur rejects. If they agree, Arthur computes $(Q(\alpha_1), \ldots, Q(\alpha_K))$ via univariate multipoint evaluation.

Batch Evaluation Protocol Proof Sketch

- The function $\Psi_j : F \to F$ is defined such that $\Psi_j(\alpha_i) = a_i[j]$.
- Each Ψ_j is a univariate polynomial of degree at most K
- Work with extension field $F = \mathbb{F}_{q^\ell}$, $q^\ell > dK/\varepsilon$. (reduces error)
- Define $R(x) := C(\Psi_1(x), \ldots, \Psi_n(x)) \in F[x]$, a univariate polynomial satisfying $R(\alpha_i) = C(a_i)$ for all i.
- Merlin sends a polynomial $Q(x)$ of degree at most dK claiming $Q(x) = R(x)$. Arthur randomly picks $r \in F$ and checks if $Q(r) = C(\Psi_1(r), \ldots, \Psi_n(r))$.
- Arthur evaluates $\Psi_j(r)$ for all j using Horner's rule, and then evaluates C on these values by simulating the arithmetic circuit over F.
- If $Q(r) \neq R(r)$, Arthur rejects. If they agree, Arthur computes $(Q(\alpha_1), \ldots, Q(\alpha_K))$ via univariate multipoint evaluation.
- The soundness follows from the Schwartz-Zippel lemma: if $Q \neq R$, then the probability $Q(r) = R(r)$ is at most $dK/q^\ell < \varepsilon$.

Arithmetic circuit evaluation

Arithmetic circuit evaluation

Given a prime p, an $\varepsilon > 0$, and an arithmetic circuit C with degree d, $s \geq n$ gates, and n variables, the sum $\sum_{(b_1,\ldots,b_n)\in\{0,1\}^n} C(b_1, \ldots, b_n) \mod p$ can be computed by a Merlin-Arthur protocol running in $2^{n/2} \cdot poly(n, s, d, \log(p/\varepsilon))$ time tossing only $n/2 + O(\log(pd/\varepsilon))$ coins, with probability of error ε.

Arithmetic circuit evaluation

Given a prime p, an $\varepsilon > 0$, and an arithmetic circuit C with degree d, $s \geq n$ gates, and n variables, the sum $\sum_{(b_1,\ldots,b_n)\in\{0,1\}^n} C(b_1,\ldots,b_n) \mod p$ can be computed by a Merlin-Arthur protocol running in $2^{n/2}\cdot poly(n,s,d,\log(p/\varepsilon))$ time tossing only $n/2 + O(\log(pd/\varepsilon))$ coins, with probability of error ε.

- Define $C'(x_1,\ldots,x_{n/2}) := \sum_{y\in\{0,1\}^{n/2}} C(x_1,\ldots,x_{n/2},y)$, a circuit of degree d and size $\leq 2^{n/2}\cdot s$.

Arithmetic circuit evaluation

Given a prime p, an $\varepsilon > 0$, and an arithmetic circuit C with degree d, $s \geq n$ gates, and n variables, the sum $\sum_{(b_1,\ldots,b_n)\in\{0,1\}^n} C(b_1,\ldots,b_n) \mod p$ can be computed by a Merlin-Arthur protocol running in $2^{n/2} \cdot poly(n, s, d, \log(p/\varepsilon))$ time tossing only $n/2 + O(\log(pd/\varepsilon))$ coins, with probability of error ε.

- Define $C'(x_1,\ldots,x_{n/2}) := \sum_{y\in\{0,1\}^{n/2}} C(x_1,\ldots,x_{n/2},y)$, a circuit of degree d and size $\leq 2^{n/2} \cdot s$.
- Use the batch evaluation MA protocol to compute C' on all $2^{n/2}$ Boolean inputs with proof size $2^{n/2} \cdot d \cdot \mathrm{poly}(n, \log(pd/\varepsilon))$ and randomness $n/2 + \log(pd/\varepsilon)$.

Arithmetic circuit evaluation

Arithmetic circuit evaluation

Given a prime p, an $\varepsilon > 0$, and an arithmetic circuit C with degree d, $s \geq n$ gates, and n variables, the sum $\sum_{(b_1,\ldots,b_n)\in\{0,1\}^n} C(b_1,\ldots,b_n) \mod p$ can be computed by a Merlin-Arthur protocol running in $2^{n/2} \cdot poly(n, s, d, \log(p/\varepsilon))$ time tossing only $n/2 + O(\log(pd/\varepsilon))$ coins, with probability of error ε.

- Define $C'(x_1,\ldots,x_{n/2}) := \sum_{y\in\{0,1\}^{n/2}} C(x_1,\ldots,x_{n/2},y)$, a circuit of degree d and size $\leq 2^{n/2} \cdot s$.
- Use the batch evaluation MA protocol to compute C' on all $2^{n/2}$ Boolean inputs with proof size $2^{n/2} \cdot d \cdot \text{poly}(n, \log(pd/\varepsilon))$ and randomness $n/2 + \log(pd/\varepsilon)$.
- Summing all $C'(x)$ gives the total $\sum_{x\in\{0,1\}^n} C(x)$ with error at most ε and total time $2^{n/2} \cdot \text{poly}(n, s, d, \log(pd/\varepsilon))$.

MA protocol for #SAT

For any $k > 0$, #SAT for Boolean formulas with n variables and m connectives has an MA-proof system using $2^{n/2} \cdot \mathrm{poly}(n, m)$ time with randomness $O(n)$ and error probability $1/\exp(n)$.

MA protocol for #SAT

MA protocol for #SAT

For any $k > 0$, #SAT for Boolean formulas with n variables and m connectives has an MA-proof system using $2^{n/2} \cdot \text{poly}(n, m)$ time with randomness $O(n)$ and error probability $1/\exp(n)$.

- Let F be a Boolean formula over AND, OR, and NOT with n variables and m connectives.

MA protocol for #SAT

MA protocol for #SAT

For any $k > 0$, #SAT for Boolean formulas with n variables and m connectives has an MA-proof system using $2^{n/2} \cdot \text{poly}(n, m)$ time with randomness $O(n)$ and error probability $1/\exp(n)$.

- Let F be a Boolean formula over AND, OR, and NOT with n variables and m connectives.
- Arithmetize formula F to a polynomial $P(x_1, \ldots, x_n)$ using: AND $\to xy$, OR $\to x + y - xy$, NOT $\to 1 - x$.

MA protocol for #SAT

MA protocol for #SAT

For any $k > 0$, #SAT for Boolean formulas with n variables and m connectives has an MA-proof system using $2^{n/2} \cdot \text{poly}(n, m)$ time with randomness $O(n)$ and error probability $1/\exp(n)$.

- Let F be a Boolean formula over AND, OR, and NOT with n variables and m connectives.
- Arithmetize formula F to a polynomial $P(x_1, \ldots, x_n)$ using: AND $\to xy$, OR $\to x + y - xy$, NOT $\to 1 - x$.
- Choose prime $p > 2^n$ (with $p < 2^{n+1}$ by Bertrand's postulate) to compute $\sum_{x \in \{0,1\}^n} P(x) \mod p$.

MA protocol for #SAT

MA protocol for #SAT

For any $k > 0$, #SAT for Boolean formulas with n variables and m connectives has an MA-proof system using $2^{n/2} \cdot \text{poly}(n, m)$ time with randomness $O(n)$ and error probability $1/\exp(n)$.

- Let F be a Boolean formula over AND, OR, and NOT with n variables and m connectives.
- Arithmetize formula F to a polynomial $P(x_1, \ldots, x_n)$ using: AND $\to xy$, OR $\to x + y - xy$, NOT $\to 1 - x$.
- Choose prime $p > 2^n$ (with $p < 2^{n+1}$ by Bertrand's postulate) to compute $\sum_{x \in \{0,1\}^n} P(x) \mod p$.
- Apply the arithmetic circuit MA protocol to compute this sum in $2^{n/2} \cdot \text{poly}(n, m)$ time with $O(n)$ randomness and error $1/\exp(n)$.

A Faster MA Protocol for K-SAT

Improvement over the Previous protocol

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.

Satisfiability Coding lemma

- Satisfying assignments can have "critical clauses" where one variable's value is forced.
- The goal is to omit these forced/determined bits.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.
- Output is the sequence of remaining bits, might be shorter.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.
- Output is the sequence of remaining bits, might be shorter.
- While decoding the sequence, process variables in the same order σ.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.
- Output is the sequence of remaining bits, might be shorter.
- While decoding the sequence, process variables in the same order σ.
- If current sequence forces $\sigma(i)$ via a unit clause, Set $\sigma(i)$ accordingly without consuming a bit.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.
- Output is the sequence of remaining bits, might be shorter.
- While decoding the sequence, process variables in the same order σ.
- If current sequence forces $\sigma(i)$ via a unit clause, Set $\sigma(i)$ accordingly without consuming a bit.
- Else Set $\sigma(i)$ using the next bit from the compressed sequence.

Satisfiability Coding lemma

- Satisfying assignments can have ”critical clauses” where one variable’s value is forced.
- The goal is to omit these forced/determined bits.
- Process variables in order of a random permutation σ.
- Delete bit for variable $\sigma(i)$ if it’s the *last* variable (per σ) in one of its critical clauses.
- Output is the sequence of remaining bits, might be shorter.
- While decoding the sequence, process variables in the same order σ.
- If current sequence forces $\sigma(i)$ via a unit clause, Set $\sigma(i)$ accordingly without consuming a bit.
- Else Set $\sigma(i)$ using the next bit from the compressed sequence.
- How short will the encoded sequences be?

Satisfiability Coding lemma

A point $x \in S$ is *"isolated"* in ith direction if flipping its i-th bit makes it leave S. It is *"j-isolated"* if it is isolated in exactly j directions.

Satisfiability Coding lemma

A point $x \in S$ is *"isolated"* in ith direction if flipping its i-th bit makes it leave S. It is *"j-isolated"* if it is isolated in exactly j directions.

Satisfiability Coding Lemma

If x is a j-isolated satisfying assignment of a k-CNF F, then its average description length under the encoding Φ_σ, is at most $n - j/k$.

Satisfiability Coding lemma

A point $x \in S$ is *"isolated"* in ith direction if flipping its i-th bit makes it leave S. It is *"j-isolated"* if it is isolated in exactly j directions.

Satisfiability Coding Lemma

If x is a j-isolated satisfying assignment of a k-CNF F, then its average description length under the encoding Φ_σ, is at most $n - j/k$.

Proof Sketch

Since x is j-isolated, j variables possess critical clauses. For a random permutation σ, each critical variable appears last in its critical clause with probability $\geq 1/k$, as clause sizes are $\leq k$, which are in turn gets deleted. The expected number of deleted bits is $\geq j/k$, resulting in a description length for x of at most $n - j/k$.

Satisfiability Coding lemma

Lemma

If $\Phi : S \rightarrow \{0,1\}^*$ *is a prefix free encoding (one-to-one function) with average code length* l*, then* $|S| \leq 2^l$.

Satisfiability Coding lemma

Lemma

If $\Phi : S \to \{0,1\}^*$ is a prefix free encoding (one-to-one function) with average code length l, then $|S| \leq 2^l$.

Let l_x denote the length of $\Phi(x)$ for $x \in S$. Then $l = \sum_{x \in S} l_x / |S|$. Since Φ is one-to-one and prefix free, we have that $\sum_{x \in S} 2^{-l_x} \leq 1$. Thus,

$$l - \log|S| = \sum_{x \in S} \frac{1}{|S|}(l_x - \log|S|) = -\sum_{x \in S} \frac{1}{|S|}(\log 2^{-l_x} + \log|S|)$$
$$= -\sum_{x \in S} \frac{1}{|S|} \log(|S| 2^{-l_x}) = -\log(\sum_{x \in S} 2^{-l_x}) \geq 0.$$

Satisfiability Coding lemma

Lemma

If $\Phi : S \to \{0,1\}^*$ is a prefix free encoding (one-to-one function) with average code length l, then $|S| \leq 2^l$.

Let l_x denote the length of $\Phi(x)$ for $x \in S$. Then $l = \sum_{x \in S} l_x / |S|$. Since Φ is one-to-one and prefix free, we have that $\sum_{x \in S} 2^{-l_x} \leq 1$. Thus,

$$l - \log|S| = \sum_{x \in S} \frac{1}{|S|}(l_x - \log|S|) = -\sum_{x \in S} \frac{1}{|S|}(\log 2^{-l_x} + \log|S|)$$
$$= -\sum_{x \in S} \frac{1}{|S|} \log(|S| 2^{-l_x}) = -\log(\sum_{x \in S} 2^{-l_x}) \geq 0.$$

The penultimate inequality follows from the concavity of the logarithm function. Hence, $|S| \leq 2^l$.

Satisfiability Coding lemma

Lemma

If $\Phi : S \to \{0,1\}^*$ is a prefix free encoding (one-to-one function) with average code length l, then $|S| \leq 2^l$.

Let l_x denote the length of $\Phi(x)$ for $x \in S$. Then $l = \sum_{x \in S} l_x / |S|$. Since Φ is one-to-one and prefix free, we have that $\sum_{x \in S} 2^{-l_x} \leq 1$. Thus,

$$l - \log|S| = \sum_{x \in S} \frac{1}{|S|}(l_x - \log|S|) = -\sum_{x \in S} \frac{1}{|S|}(\log 2^{-l_x} + \log|S|)$$
$$= -\sum_{x \in S} \frac{1}{|S|} \log(|S| 2^{-l_x}) = -\log(\sum_{x \in S} 2^{-l_x}) \geq 0.$$

The penultimate inequality follows from the concavity of the logarithm function. Hence, $|S| \leq 2^l$.
Corollary. Any k-CNF F can accept at most $2^{n-n/k}$ isolated solutions.

Variable Reduction Lemma

We will first state the *"Sparsification Lemma"*:

Variable Reduction Lemma

We will first state the *"Sparsification Lemma"*:

Sparsification Lemma

For all $\varepsilon > 0$, k-CNF F can be written as the disjunction of at most $2^{\varepsilon n}$ k-CNF F_i such that F_i contains each variable in at most $c(k, \varepsilon)$ clauses for some function c. Moreover, this reduction takes at most $\text{poly}(n)2^{\varepsilon n}$ time.

Variable Reduction Lemma

We will first state the *"Sparsification Lemma"*:

Sparsification Lemma

For all $\varepsilon > 0$, k-CNF F can be written as the disjunction of at most $2^{\varepsilon n}$ k-CNF F_i such that F_i contains each variable in at most $c(k, \varepsilon)$ clauses for some function c. Moreover, this reduction takes at most $\text{poly}(n)2^{\varepsilon n}$ time.

Next, we state the *"Variable Reduction Lemma"*:

Variable Reduction Lemma

We will first state the *"Sparsification Lemma"*:

Sparsification Lemma

For all $\varepsilon > 0$, k-CNF F can be written as the disjunction of at most $2^{\varepsilon n}$ k-CNF F_i such that F_i contains each variable in at most $c(k, \varepsilon)$ clauses for some function c. Moreover, this reduction takes at most $\text{poly}(n)2^{\varepsilon n}$ time.

Next, we state the *"Variable Reduction Lemma"*:

Variable Reduction Lemma

Let F be a k-CNF formula on m clauses such that every satisfying assignment to F has at least δn variables set to true for any $\delta > 0$. For any $\varepsilon > 0$, there exists a $k' > 0$ and F', which is a disjunction of at most $2^{\varepsilon n}$ k'-CNFs on at most $n(1 - \delta/(ek))$ variables such that F is satisfiable iff F' is satisfiable. Moreover F' can be computed from F in $2^{2\varepsilon n} \cdot \text{poly}(m)$ time.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.
- Trade-off, increase the clause width, reduce number of variables.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.
- Trade-off, increase the clause width, reduce number of variables.
- Let A, B be a random partition, for each variable x, $x \in B$ with probability $1/k$, otherwise $x \in A$.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.
- Trade-off, increase the clause width, reduce number of variables.
- Let A, B be a random partition, for each variable x, $x \in B$ with probability $1/k$, otherwise $x \in A$.
- Then B contains at least $n/(ek)$ forced variables on average with respect to the assignment α_A, the restriction of α to A.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.
- Trade-off, increase the clause width, reduce number of variables.
- Let A, B be a random partition, for each variable x, $x \in B$ with probability $1/k$, otherwise $x \in A$.
- Then B contains at least $n/(ek)$ forced variables on average with respect to the assignment α_A, the restriction of α to A.
- Want to eliminate these forced variables by rewriting them in terms of other variables.

Proof sketch

Let's first prove it for UniqueSAT instance, assume sparseness.

- Let F be a k-CNF with at most one satisfying assignment (denote by α, if exixts) and each variable appearing in at most c clauses.
- α being unique satisfying assignment, all n variables are **critical**.
- Goal is to eliminate the forced variables by rewriting them in terms of other variables.
- Trade-off, increase the clause width, reduce number of variables.
- Let A, B be a random partition, for each variable x, $x \in B$ with probability $1/k$, otherwise $x \in A$.
- Then B contains at least $n/(ek)$ forced variables on average with respect to the assignment α_A, the restriction of α to A.
- Want to eliminate these forced variables by rewriting them in terms of other variables.
- Want to find a formula $F(A, B)$ which only depend on the variables in A and the unforced variables of B such that $F(A, B)$ is satisfiable iff F is satisfiable.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.
- A clause C is a **positive** (x, A) clause if it contains x (not $\bar{x}$) and all other variables in A.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.
- A clause C is a **positive** (x, A) clause if it contains x (not $\bar{x}$) and all other variables in A.
- If an (x, A) clause is critical for x at α, then all other literals in C are false under α_A.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.
- A clause C is a **positive** (x, A) clause if it contains x (not $\bar{x}$) and all other variables in A.
- If an (x, A) clause is critical for x at α, then all other literals in C are false under α_A.
- G_x is the disjunction over all (x, A) clauses of terms formed by negating all literals in the clause except x or $\bar{x}$.

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.
- A clause C is a **positive** (x, A) clause if it contains x (not $\bar{x}$) and all other variables in A.
- If an (x, A) clause is critical for x at α, then all other literals in C are false under α_A.
- G_x is the disjunction over all (x, A) clauses of terms formed by negating all literals in the clause except x or $\bar{x}$.
- Similarly, G'_x is the disjunction over all positive (x, A) clauses of terms formed by negating all literals except x. It expresses "x is forced to be true."

Proof Contd.

- For each $x \in B$, the proposition "x is forced by α" is expressed by a DNF formula G_x, with at most c terms, each term containing at most $(k-1)$ literals.
- A clause C of F is an (x, A) clause if x or $\bar{x}$ occurs in C, and all other variables in C are from A.
- A clause C is a **positive** (x, A) clause if it contains x (not $\bar{x}$) and all other variables in A.
- If an (x, A) clause is critical for x at α, then all other literals in C are false under α_A.
- G_x is the disjunction over all (x, A) clauses of terms formed by negating all literals in the clause except x or $\bar{x}$.
- Similarly, G'_x is the disjunction over all positive (x, A) clauses of terms formed by negating all literals except x. It expresses "x is forced to be true."
- Both G_x and G'_x depend on at most $c(k-1)$ variables in A.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).
- Goal: eliminate forced variables in B_i and rename unforced ones using new variables $Y_i = \{y_{i,1}, \ldots, y_{i,l-f_i}\}$.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).
- Goal: eliminate forced variables in B_i and rename unforced ones using new variables $Y_i = \{y_{i,1}, \ldots, y_{i,l-f_i}\}$.
- Define slice functions $\Phi_{i,j}$ that evaluate true iff exactly j variables in B_i are true; they depend only on variables in A and at most $cl(k-1)$ of them.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).
- Goal: eliminate forced variables in B_i and rename unforced ones using new variables $Y_i = \{y_{i,1}, \ldots, y_{i,l-f_i}\}$.
- Define slice functions $\Phi_{i,j}$ that evaluate true iff exactly j variables in B_i are true; they depend only on variables in A and at most $cl(k-1)$ of them.
- For each $x_j \in B_i$, determine if it is forced. If not, assign it to the j'th unforced variable renamed as $y_{i,j}$.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).
- Goal: eliminate forced variables in B_i and rename unforced ones using new variables $Y_i = \{y_{i,1}, \ldots, y_{i,l-f_i}\}$.
- Define slice functions $\Phi_{i,j}$ that evaluate true iff exactly j variables in B_i are true; they depend only on variables in A and at most $cl(k-1)$ of them.
- For each $x_j \in B_i$, determine if it is forced. If not, assign it to the j'th unforced variable renamed as $y_{i,j}$.
- Construct Boolean expression $\beta_j(G_{x_1}, \ldots, G_{x_{j-1}}, y_{i,1}, \ldots, y_{i,j})$ to select the correct $y_{i,q}$ based on number of previous forced variables.

Proof Contd.

- Let l be a parameter to be fixed later; partition B arbitrarily into $B_1, \ldots, B_p$ of size l.
- For each B_i, let f_i be the number of forced variables (determined by partial assignment to A).
- Goal: eliminate forced variables in B_i and rename unforced ones using new variables $Y_i = \{y_{i,1}, \ldots, y_{i,l-f_i}\}$.
- Define slice functions $\Phi_{i,j}$ that evaluate true iff exactly j variables in B_i are true; they depend only on variables in A and at most $cl(k-1)$ of them.
- For each $x_j \in B_i$, determine if it is forced. If not, assign it to the j'th unforced variable renamed as $y_{i,j}$.
- Construct Boolean expression $\beta_j(G_{x_1}, \ldots, G_{x_{j-1}}, y_{i,1}, \ldots, y_{i,j})$ to select the correct $y_{i,q}$ based on number of previous forced variables.
- Define $\Psi_{i,x_j} := G'_{x_j} \vee (\overline{G_{x_j}} \wedge \beta_j)$ to express whether x_j is forced to be true or corresponds to some $y_{i,q}$.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.
- Substitute each $x_j \in B_i$ with Ψ_{i,x_j} in F to obtain a new formula F' over $Y = A \cup \bigcup_i Y_i$.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.
- Substitute each $x_j \in B_i$ with Ψ_{i,x_j} in F to obtain a new formula F' over $Y = A \cup \bigcup_i Y_i$.
- After substitution, each clause in F' depends on at most lck^2 variables; define $k' = clk^2$.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.
- Substitute each $x_j \in B_i$ with Ψ_{i,x_j} in F to obtain a new formula F' over $Y = A \cup \bigcup_i Y_i$.
- After substitution, each clause in F' depends on at most lck^2 variables; define $k' = clk^2$.
- Define $\Gamma_{\vec{f}} := F' \wedge \bigwedge_{i=1}^{p} \Phi_{i,f_i}$ to encode the choice of forced variable counts f_i.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.
- Substitute each $x_j \in B_i$ with Ψ_{i,x_j} in F to obtain a new formula F' over $Y = A \cup \bigcup_i Y_i$.
- After substitution, each clause in F' depends on at most lck^2 variables; define $k' = clk^2$.
- Define $\Gamma_{\vec{f}} := F' \wedge \bigwedge_{i=1}^{p} \Phi_{i,f_i}$ to encode the choice of forced variable counts f_i.
- By defining $\Gamma_{\vec{f}} := F' \wedge \bigwedge_{i=1}^{p} \Phi_{i,f_i}$ and choosing l such that $\frac{\log(l+1)}{l} \leq \varepsilon$, the number of such vectors $\vec{f}$ becomes at most $2^{\varepsilon n}$. Hence, $\Gamma := \bigvee_{\vec{f}} \Gamma_{\vec{f}}$ is a disjunction of at most $2^{\varepsilon n}$ k'-CNF formulas, each on at most $n(1-1/(ek))$ variables, with $k' = clk^2$.

Proof Contd.

- Ψ_{i,x_j} depends on at most $lc(k-1)$ variables from A and Y_i, hence total lck variables.
- Substitute each $x_j \in B_i$ with Ψ_{i,x_j} in F to obtain a new formula F' over $Y = A \cup \bigcup_i Y_i$.
- After substitution, each clause in F' depends on at most lck^2 variables; define $k' = clk^2$.
- Define $\Gamma_{\vec{f}} := F' \wedge \bigwedge_{i=1}^{p} \Phi_{i,f_i}$ to encode the choice of forced variable counts f_i.
- By defining $\Gamma_{\vec{f}} := F' \wedge \bigwedge_{i=1}^{p} \Phi_{i,f_i}$ and choosing l such that $\frac{\log(l+1)}{l} \leq \varepsilon$, the number of such vectors $\vec{f}$ becomes at most $2^{\varepsilon n}$. Hence, $\Gamma := \bigvee_{\vec{f}} \Gamma_{\vec{f}}$ is a disjunction of at most $2^{\varepsilon n}$ k'-CNF formulas, each on at most $n(1-1/(ek))$ variables, with $k' = clk^2$.
- If F is uniquely satisfiable, exactly one $\Gamma_{\vec{f}}$ is uniquely satisfiable; if F is unsatisfiable, all $\Gamma_{\vec{f}}$ are unsatisfiable, so Γ is also unsatisfiable.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.
- For each such partition, we construct Γ_{AB} and define $\Gamma = \bigvee_{A,B} \Gamma_{AB}$ as the disjunction over all such partitions.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.
- For each such partition, we construct Γ_{AB} and define $\Gamma = \bigvee_{A,B} \Gamma_{AB}$ as the disjunction over all such partitions.
- Initially, we assumed that each variable in F appears in at most c clauses.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.
- For each such partition, we construct Γ_{AB} and define $\Gamma = \bigvee_{A,B} \Gamma_{AB}$ as the disjunction over all such partitions.
- Initially, we assumed that each variable in F appears in at most c clauses.
- If this assumption does not hold, apply the Sparsification Lemma to express $F = \bigvee_i F_i$ such that each F_i has at most $c(k, \varepsilon)$ occurrences per variable and there are at most $2^{\varepsilon n}$ such F_i.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.
- For each such partition, we construct Γ_{AB} and define $\Gamma = \bigvee_{A,B} \Gamma_{AB}$ as the disjunction over all such partitions.
- Initially, we assumed that each variable in F appears in at most c clauses.
- If this assumption does not hold, apply the Sparsification Lemma to express $F = \bigvee_i F_i$ such that each F_i has at most $c(k, \varepsilon)$ occurrences per variable and there are at most $2^{\varepsilon n}$ such F_i.
- For each F_i, construct Γ_i using the previous method, and define the final formula $\Gamma = \bigvee_i \Gamma_i$.

Proof Contd.

- To remove randomness in partitioning variables into A and B, we try all partitions (A, B) from a k-wise independent probability space of size $n^{O(k)}$.
- For each such partition, we construct Γ_{AB} and define $\Gamma = \bigvee_{A,B} \Gamma_{AB}$ as the disjunction over all such partitions.
- Initially, we assumed that each variable in F appears in at most c clauses.
- If this assumption does not hold, apply the Sparsification Lemma to express $F = \bigvee_i F_i$ such that each F_i has at most $c(k, \varepsilon)$ occurrences per variable and there are at most $2^{\varepsilon n}$ such F_i.
- For each F_i, construct Γ_i using the previous method, and define the final formula $\Gamma = \bigvee_i \Gamma_i$.
- Since each Γ_i is a disjunction of at most $2^{\varepsilon n}$ k'-CNFs, the total Γ is a disjunction of at most $2^{2\varepsilon n}$ k'-CNFs.

Proof Contd.

Lemma

Let F be a k-CNF such that F is not satisfiable by any assignment that contains fewer than δn 1's. For any $\varepsilon > 0$, there exists k' such that the following holds: The satisfiability of F is equivalent to the satisfiability of $\hat{F}$ where $\hat{F}$ is a disjunction of at most $2^{2\varepsilon n}$ k'-CNFs on at most $n(1 - \delta/(ek))$ variables. Moreover, $\hat{F}$ can be computed from F in time $\mathrm{poly}(n)2^{2\varepsilon n}$.

Proof Contd.

Lemma

Let F be a k-CNF such that F is not satisfiable by any assignment that contains fewer than δn 1's. For any $\varepsilon > 0$, there exists k' such that the following holds: The satisfiability of F is equivalent to the satisfiability of $\hat{F}$ where $\hat{F}$ is a disjunction of at most $2^{2\varepsilon n}$ k'-CNFs on at most $n(1 - \delta/(ek))$ variables. Moreover, $\hat{F}$ can be computed from F in time $\mathrm{poly}(n)2^{2\varepsilon n}$.

- For general k-SAT, consider α to be δn-isolated satisfying assignment which is also minimal.

Proof Contd.

Lemma

Let F be a k-CNF such that F is not satisfiable by any assignment that contains fewer than δn 1's. For any $\varepsilon > 0$, there exists k' such that the following holds: The satisfiability of F is equivalent to the satisfiability of $\hat{F}$ where $\hat{F}$ is a disjunction of at most $2^{2\varepsilon n}$ k'-CNFs on at most $n(1 - \delta/(ek))$ variables. Moreover, $\hat{F}$ can be computed from F in time $\mathrm{poly}(n)2^{2\varepsilon n}$.

- For general k-SAT, consider α to be δn-isolated satisfying assignment which is also minimal.
- α has at least δn 1's and by minimality α is isolated with respect to each of these δn variables.

Proof Contd.

Lemma

Let F be a k-CNF such that F is not satisfiable by any assignment that contains fewer than δn 1's. For any $\varepsilon > 0$, there exists k' such that the following holds: The satisfiability of F is equivalent to the satisfiability of $\hat{F}$ where $\hat{F}$ is a disjunction of at most $2^{2\varepsilon n}$ k'-CNFs on at most $n(1 - \delta/(ek))$ variables. Moreover, $\hat{F}$ can be computed from F in time $\mathrm{poly}(n)2^{2\varepsilon n}$.

- For general k-SAT, consider α to be δn-isolated satisfying assignment which is also minimal.
- α has at least δn 1's and by minimality α is isolated with respect to each of these δn variables.
- partition A and B will on average force at least $\delta n/(ek)$ variables in B with respect to α_A. Rest of the proof is similar to the UniqueSAT case.

A Faster Protocol

Theorem

There is a universal constant $\delta > 0$ such that for all sufficiently large integers $k > 0$, we can verify unsatisfiable n-variable m-clause k-CNF with a Merlin-Arthur protocol running in $2^{n(1/2-\delta/k)} \cdot poly(n, m)$ time.

A Faster Protocol

Theorem

There is a universal constant $\delta > 0$ such that for all sufficiently large integers $k > 0$, we can verify unsatisfiable n-variable m-clause k-CNF with a Merlin-Arthur protocol running in $2^{n(1/2-\delta/k)} \cdot \text{poly}(n, m)$ time.

Note that, this protocol improves on the earlier $2^{n/2} \cdot \text{poly}(n, m)$ bound. I will next present the proof outline.

A Faster Protocol

Theorem

There is a universal constant $\delta > 0$ such that for all sufficiently large integers $k > 0$, we can verify unsatisfiable n-variable m-clause k-CNF with a Merlin-Arthur protocol running in $2^{n(1/2-\delta/k)} \cdot poly(n, m)$ time.

Note that, this protocol improves on the earlier $2^{n/2} \cdot \text{poly}(n, m)$ bound. I will next present the proof outline.

- Given a k-CNF formula F with n variables and m clauses, Arthur and Merlin aim to certify F is unsatisfiable.

A Faster Protocol

Theorem

There is a universal constant $\delta > 0$ such that for all sufficiently large integers $k > 0$, we can verify unsatisfiable n-variable m-clause k-CNF with a Merlin-Arthur protocol running in $2^{n(1/2-\delta/k)} \cdot poly(n, m)$ time.

Note that, this protocol improves on the earlier $2^{n/2} \cdot \text{poly}(n, m)$ bound. I will next present the proof outline.

- Given a k-CNF formula F with n variables and m clauses, Arthur and Merlin aim to certify F is unsatisfiable.
- Arthur enumerates all assignments with at most δn variables set to true and checks none satisfy F — this takes time $2^{H(\delta)n} \cdot \text{poly}(n, m)$.

A Faster Protocol

Theorem

There is a universal constant $\delta > 0$ such that for all sufficiently large integers $k > 0$, we can verify unsatisfiable n-variable m-clause k-CNF with a Merlin-Arthur protocol running in $2^{n(1/2-\delta/k)} \cdot poly(n, m)$ time.

Note that, this protocol improves on the earlier $2^{n/2} \cdot \text{poly}(n, m)$ bound. I will next present the proof outline.

- Given a k-CNF formula F with n variables and m clauses, Arthur and Merlin aim to certify F is unsatisfiable.
- Arthur enumerates all assignments with at most δn variables set to true and checks none satisfy F — this takes time $2^{H(\delta)n} \cdot \text{poly}(n, m)$.
- Next, apply the above theorem with $\varepsilon = 1/k^2$ to transform F into $t = 2^{n/k^2}$ k'-CNFs: $F'_1, \ldots, F'_t$.

Proof Sketch

- Each F_i' has only $n(1-\delta/(ek))$ variables, significantly reducing the variable count.

Proof Sketch

- Each F_i' has only $n(1 - \delta/(ek))$ variables, significantly reducing the variable count.
- For each F_i', Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.

Proof Sketch

- Each F_i' has only $n(1 - \delta/(ek))$ variables, significantly reducing the variable count.
- For each F_i', Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.
- Each verification takes $2^{n(1/2-\delta/(2ek))} \cdot \text{poly}(n, m)$ time.

Proof Sketch

- Each F_i' has only $n(1-\delta/(ek))$ variables, significantly reducing the variable count.
- For each F_i', Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.
- Each verification takes $2^{n(1/2-\delta/(2ek))} \cdot \mathrm{poly}(n,m)$ time.
- Total verification time across all F_i' is:

$$2^{n/k^2} \cdot 2^{n(1/2-\delta/(2ek))} = 2^{n(1/2-\delta/(6k))} \cdot \mathrm{poly}(n,m)$$

  for large enough k (e.g., $k \geq 60$).

Proof Sketch

- Each F_i' has only $n(1-\delta/(ek))$ variables, significantly reducing the variable count.
- For each F_i', Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.
- Each verification takes $2^{n(1/2-\delta/(2ek))} \cdot \mathrm{poly}(n,m)$ time.
- Total verification time across all F_i' is:

$$2^{n/k^2} \cdot 2^{n(1/2-\delta/(2ek))} = 2^{n(1/2-\delta/(6k))} \cdot \mathrm{poly}(n,m)$$

  for large enough k (e.g., $k \geq 60$).
- Hence, the total running time of the protocol is: $(2^{n(1/2-\delta/(6k))} + 2^{H(\delta)n}) \cdot \mathrm{poly}(n,m)$.

Proof Sketch

- Each F'_i has only $n(1 - \delta/(ek))$ variables, significantly reducing the variable count.
- For each F'_i, Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.
- Each verification takes $2^{n(1/2-\delta/(2ek))} \cdot \text{poly}(n, m)$ time.
- Total verification time across all F'_i is:

$$2^{n/k^2} \cdot 2^{n(1/2-\delta/(2ek))} = 2^{n(1/2-\delta/(6k))} \cdot \text{poly}(n, m)$$

  for large enough k (e.g., $k \geq 60$).
- Hence, the total running time of the protocol is: $(2^{n(1/2-\delta/(6k))} + 2^{H(\delta)n}) \cdot \text{poly}(n, m)$.
- Set δ small enough such that $H(\delta) \leq 2\delta \log_2(1/\delta) \leq \frac{1}{2} - \frac{\delta}{k}$.

Proof Sketch

- Each F'_i has only $n(1 - \delta/(ek))$ variables, significantly reducing the variable count.
- For each F'_i, Merlin sends a proof and Arthur verifies its unsatisfiability using the Batch Evaluation protocol.
- Each verification takes $2^{n(1/2-\delta/(2ek))} \cdot \mathrm{poly}(n, m)$ time.
- Total verification time across all F'_i is:

$$2^{n/k^2} \cdot 2^{n(1/2-\delta/(2ek))} = 2^{n(1/2-\delta/(6k))} \cdot \mathrm{poly}(n, m)$$

  for large enough k (e.g., $k \geq 60$).
- Hence, the total running time of the protocol is: $(2^{n(1/2-\delta/(6k))} + 2^{H(\delta)n}) \cdot \mathrm{poly}(n, m)$.
- Set δ small enough such that $H(\delta) \leq 2\delta \log_2(1/\delta) \leq \frac{1}{2} - \frac{\delta}{k}$.
- This completes the proof of the improved Merlin-Arthur protocol for k-UNSAT.

Our Results

MA Protocol For SUB-SAT and POLY-EQS Problem

SUB-SAT problem

- Given an n-variate Boolean formula Φ and an affine subspace $A \subseteq \mathbb{F}_2^n$ (described by a system of $\mathbb{F}_2$-linear equations), we aim to design a Merlin-Arthur (MA) protocol to decide if Φ has a satisfying assignment in A.

SUB-SAT problem

- Given an n-variate Boolean formula Φ and an affine subspace $A \subseteq \mathbb{F}_2^n$ (described by a system of $\mathbb{F}_2$-linear equations), we aim to design a Merlin-Arthur (MA) protocol to decide if Φ has a satisfying assignment in A.
- We refer to this problem as *satisfiability in a subspace*, abbreviated as **SUB-SAT**. It generalizes the standard SAT problem by incorporating a linear-algebraic constraint on the space of assignments.

SUB-SAT problem

- Given an n-variate Boolean formula Φ and an affine subspace $A \subseteq \mathbb{F}_2^n$ (described by a system of $\mathbb{F}_2$-linear equations), we aim to design a Merlin-Arthur (MA) protocol to decide if Φ has a satisfying assignment in A.
- We refer to this problem as *satisfiability in a subspace*, abbreviated as **SUB-SAT**. It generalizes the standard SAT problem by incorporating a linear-algebraic constraint on the space of assignments.
- Since SUB-SAT generalizes SAT, it inherits the computational hardness and intractability associated with SAT.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.
- Arithmetize Φ using standard rules: OR $\to x + y - xy$, AND $\to xy$, NOT $\to 1 - x$, yielding an arithmetic formula P.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.
- Arithmetize Φ using standard rules: OR $\rightarrow x + y - xy$, AND $\rightarrow xy$, NOT $\rightarrow 1 - x$, yielding an arithmetic formula P.
- Each affine constraint like $x_1 \oplus x_2 \oplus x_3 = 1$ is interpreted algebraically: sum is odd $\Leftrightarrow$ equation is satisfied. Adjust RHS to 1 as needed.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.
- Arithmetize Φ using standard rules: OR $\to x + y - xy$, AND $\to xy$, NOT $\to 1 - x$, yielding an arithmetic formula P.
- Each affine constraint like $x_1 \oplus x_2 \oplus x_3 = 1$ is interpreted algebraically: sum is odd $\Leftrightarrow$ equation is satisfied. Adjust RHS to 1 as needed.
- Let L be the product of LHS expressions of all affine constraints (adjusted to have RHS = 1). Define $P' = P \cdot L$. Then P' is odd iff assignment satisfies both Φ and A.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.
- Arithmetize Φ using standard rules: OR $\to x + y - xy$, AND $\to xy$, NOT $\to 1 - x$, yielding an arithmetic formula P.
- Each affine constraint like $x_1 \oplus x_2 \oplus x_3 = 1$ is interpreted algebraically: sum is odd $\Leftrightarrow$ equation is satisfied. Adjust RHS to 1 as needed.
- Let L be the product of LHS expressions of all affine constraints (adjusted to have RHS = 1). Define $P' = P \cdot L$. Then P' is odd iff assignment satisfies both Φ and A.
- But, summing P' over all assignments doesn't reveal if any term is odd, as an even number of odd terms gives an even sum — hence direct summation fails to detect satisfiability.

MA Protocol for SUB-SAT

- We will first arithmetize the instance.
- Input: a Boolean formula Φ on n variables and an affine subspace $A \subseteq \mathbb{F}_2^n$ given by $\mathbb{F}_2$-linear equations.
- Arithmetize Φ using standard rules: OR $\to x + y - xy$, AND $\to xy$, NOT $\to 1 - x$, yielding an arithmetic formula P.
- Each affine constraint like $x_1 \oplus x_2 \oplus x_3 = 1$ is interpreted algebraically: sum is odd $\Leftrightarrow$ equation is satisfied. Adjust RHS to 1 as needed.
- Let L be the product of LHS expressions of all affine constraints (adjusted to have RHS = 1). Define $P' = P \cdot L$. Then P' is odd iff assignment satisfies both Φ and A.
- But, summing P' over all assignments doesn't reveal if any term is odd, as an even number of odd terms gives an even sum — hence direct summation fails to detect satisfiability.
- Even number of SAT assignments is a problem, can make it odd?

MA Protocol for SUB-SAT Continued

Yes, using Valiant-Vazirani Lemma. We will first state the lemma:

MA Protocol for SUB-SAT Continued

Yes, using Valiant-Vazirani Lemma. We will first state the lemma:

Valiant-Vazirani Lemma

Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,

$$\Pr_{h \in \mathcal{H}_{n,k}} \left[\left| \left\{ x \in S : h(x) = 0^k \right\} \right| = 1 \right] \geq \frac{1}{8}$$

MA Protocol for SUB-SAT Continued

Yes, using Valiant-Vazirani Lemma. We will first state the lemma:

Valiant-Vazirani Lemma

Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,

$$\Pr_{h \in \mathcal{H}_{n,k}} \left[\left| \left\{ x \in S : h(x) = 0^k \right\} \right| = 1 \right] \geq \frac{1}{8}$$

- Intend to reduce the number of SAT assignments to 1 (odd).

MA Protocol for SUB-SAT Continued

Yes, using Valiant-Vazirani Lemma. We will first state the lemma:

Valiant-Vazirani Lemma

Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,

$$\Pr_{h \in \mathcal{H}_{n,k}} \left[\left| \left\{ x \in S : h(x) = 0^k \right\} \right| = 1 \right] \geq \frac{1}{8}$$

- Intend to reduce the number of SAT assignments to 1 (odd).
- Randomly guess k such that the number of satisfying assignments lies between 2^k and 2^{k+1}.

MA Protocol for SUB-SAT Continued

Yes, using Valiant-Vazirani Lemma. We will first state the lemma:

Valiant-Vazirani Lemma

Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,

$$\Pr_{h \in \mathcal{H}_{n,k}} \left[\left| \left\{ x \in S : h(x) = 0^k \right\} \right| = 1 \right] \geq \frac{1}{8}$$

- Intend to reduce the number of SAT assignments to 1 (odd).
- Randomly guess k such that the number of satisfying assignments lies between 2^k and 2^{k+1}.
- Choose random vectors $a_1, \ldots, a_{k+2} \in \{0,1\}^n$ and bits $b_1 = \cdots = b_{k+2} = 1$; define

$$P''(x) = P'(x) \cdot \prod_{i=1}^{k+2} (a_i \cdot x + b_i)$$

MA Protocol for SUB-SAT Continued

- With probability $\geq 1/8n$, P'' evaluates to an odd number on exactly one satisfying assignment (if any exist).

MA Protocol for SUB-SAT Continued

- With probability $\geq 1/8n$, P'' evaluates to an odd number on exactly one satisfying assignment (if any exist).
- Define $P'''(x_1, \ldots, x_{n/2})$ by summing P'' over $x_{n/2+1}, \ldots, x_n \in \{0,1\}^{n/2}$:

$$P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n} P''(x_1, \ldots, x_n)$$

MA Protocol for SUB-SAT Continued

- With probability $\geq 1/8n$, P'' evaluates to an odd number on exactly one satisfying assignment (if any exist).
- Define $P'''(x_1, \ldots, x_{n/2})$ by summing P'' over $x_{n/2+1}, \ldots, x_n \in \{0, 1\}^{n/2}$:

$$P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n} P''(x_1, \ldots, x_n)$$

- Use batch evaluation to check P''' on $2^{n/2}$ points. If Φ is satisfiable, one evaluation is odd with probability $1/\text{poly}(n)$; otherwise, all outputs are even.

MA Protocol for SUB-SAT Continued

- With probability $\geq 1/8n$, P'' evaluates to an odd number on exactly one satisfying assignment (if any exist).
- Define $P'''(x_1, \ldots, x_{n/2})$ by summing P'' over $x_{n/2+1}, \ldots, x_n \in \{0, 1\}^{n/2}$:

$$P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n} P''(x_1, \ldots, x_n)$$

- Use batch evaluation to check P''' on $2^{n/2}$ points. If Φ is satisfiable, one evaluation is odd with probability $1/\text{poly}(n)$; otherwise, all outputs are even.
- Repeat the procedure $64n^2$ times, if the original SUB-SAT instance is satisfiable, the algorithm will say **Yes** with probability $1 - \left(1 - \left(\frac{1}{8n}\right)\right)^{64n^2} \approx 1 - e^{-n}$, because each reduction attempt is independent of one another and $\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$

MA protocol for POLY-EQS

- In k-POLY-EQS, each P_i has degree at most k. Since $x^2 = x$ in $\mathbb{F}_2$, we can multi-linearize all equations.

MA protocol for POLY-EQS

- In k-POLY-EQS, each P_i has degree at most k. Since $x^2 = x$ in $\mathbb{F}_2$, we can multi-linearize all equations.
- k-POLY-EQS generalizes k-SUB-SAT: each clause becomes a polynomial $P_i = \prod_{j=1}^{k}(\ell_{i,j} + 1)$ such that $P_i(x) = 0$ iff the clause is satisfied.

MA protocol for POLY-EQS

- In k-POLY-EQS, each P_i has degree at most k. Since $x^2 = x$ in $\mathbb{F}_2$, we can multi-linearize all equations.
- k-POLY-EQS generalizes k-SUB-SAT: each clause becomes a polynomial $P_i = \prod_{j=1}^{k}(\ell_{i,j} + 1)$ such that $P_i(x) = 0$ iff the clause is satisfied.
- Unlike SUB-SAT, there's no Boolean formula Φ; we directly arithmetize the polynomials and multiply their LHS to form P'.

MA protocol for POLY-EQS

- In k-POLY-EQS, each P_i has degree at most k. Since $x^2 = x$ in $\mathbb{F}_2$, we can multi-linearize all equations.
- k-POLY-EQS generalizes k-SUB-SAT: each clause becomes a polynomial $P_i = \prod_{j=1}^{k}(\ell_{i,j} + 1)$ such that $P_i(x) = 0$ iff the clause is satisfied.
- Unlike SUB-SAT, there's no Boolean formula Φ; we directly arithmetize the polynomials and multiply their LHS to form P'.
- Let $P'(x) = \prod_{i=1}^{m}(P_i(x) + c_i)$ where c_i is 1 if RHS was 1. Degree of P' is at most nl.

MA protocol for POLY-EQS

- In k-POLY-EQS, each P_i has degree at most k. Since $x^2 = x$ in $\mathbb{F}_2$, we can multi-linearize all equations.
- k-POLY-EQS generalizes k-SUB-SAT: each clause becomes a polynomial $P_i = \prod_{j=1}^{k}(\ell_{i,j} + 1)$ such that $P_i(x) = 0$ iff the clause is satisfied.
- Unlike SUB-SAT, there's no Boolean formula Φ; we directly arithmetize the polynomials and multiply their LHS to form P'.
- Let $P'(x) = \prod_{i=1}^{m}(P_i(x) + c_i)$ where c_i is 1 if RHS was 1. Degree of P' is at most nl.
- Choose prime $p > 2^n n^l$ to bound all intermediate evaluations.

MA protocol for POLY-EQS Continued

- Repeat for $64n^2$ rounds: pick $k \in \{0, \dots, n-1\}$, random $a_1, \dots, a_{k+2} \in \{0,1\}^n$, and define

$$P''(x) = P'(x) \cdot \prod_{i=1}^{k+2} (a_i \cdot x + 1)$$

MA protocol for POLY-EQS Continued

- Repeat for $64n^2$ rounds: pick $k \in \{0, \ldots, n-1\}$, random $a_1, \ldots, a_{k+2} \in \{0,1\}^n$, and define

$$P''(x) = P'(x) \cdot \prod_{i=1}^{k+2} (a_i \cdot x + 1)$$

- Define $P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n \in \{0,1\}} P''(x_1, \ldots, x_n)$.

MA protocol for POLY-EQS Continued

- Repeat for $64n^2$ rounds: pick $k \in \{0, \ldots, n-1\}$, random $a_1, \ldots, a_{k+2} \in \{0,1\}^n$, and define

$$P''(x) = P'(x) \cdot \prod_{i=1}^{k+2} (a_i \cdot x + 1)$$

- Define $P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n \in \{0,1\}} P''(x_1, \ldots, x_n)$.
- Use WILLIAM'S BATCH EVALUATION on $2^{n/2}$ points; if any P''' evaluates to an odd value, declare the instance satisfiable; else, unsatisfiable.

MA protocol for POLY-EQS Continued

- Repeat for $64n^2$ rounds: pick $k \in \{0, \ldots, n-1\}$, random $a_1, \ldots, a_{k+2} \in \{0,1\}^n$, and define

$$P''(x) = P'(x) \cdot \prod_{i=1}^{k+2} (a_i \cdot x + 1)$$

- Define $P'''(x_1, \ldots, x_{n/2}) = \sum_{x_{n/2+1}, \ldots, x_n \in \{0,1\}} P''(x_1, \ldots, x_n)$.
- Use WILLIAM'S BATCH EVALUATION on $2^{n/2}$ points; if any P''' evaluates to an odd value, declare the instance satisfiable; else, unsatisfiable.
- Following a similar analysis to SUB-SAT, we can say that if the POLY-EQS instance is not satisfiable, then the protocol will always return UNSAT. On the other hand, if the POLY-EQS instance is satisfiable, with probability $1 - 1/exp(n)$, the protocol will return SAT.

Thank You