Computation That Counts

Xiaofeng Gu[1]

[1] Math Department
West Virginia University

May 1, 2009

Outline

Outline

Outline

Introduction

Problems

1. Decision problems: whether a solution exists.
2. Function (search) problems: find a solution.
3. Counting problems: how many solutions exists

Examples

1. #SAT: Given a Boolean expression, compute the number of different assignments that satisfy it.
2. #HAMILTON PATH: compute the number of different Hamilton path in a given graph.

Introduction

Problems

1. Decision problems: whether a solution exists.
2. Function (search) problems: find a solution.
3. Counting problems: how many solutions exists

Examples

1. #SAT: Given a Boolean expression, compute the number of different assignments that satisfy it.
2. #HAMILTON PATH: compute the number of different Hamilton path in a given graph.

Introduction

Problems

1. Decision problems: whether a solution exists.
2. Function (search) problems: find a solution.
3. Counting problems: how many solutions exists

Examples

1. #SAT: Given a Boolean expression, compute the number of different assignments that satisfy it.
2. #HAMILTON PATH: compute the number of different Hamilton path in a given graph.

Properties

Note

1. Counting problems cannot be easier than their decision versions. The decision problem has a solution if and only if the solution number is larger than 0.
2. Counting problems might be very hard even the decision versions is in P. For example, CYCLE asks if a directed graph contains a cycle, and it is in P. #CYCLE counts the number of cycles in a directed graph. #CYCLE is hard.

Permanent

MATCHING

1. MATCHING: Whether a bipartile graph has a perfect matching?
2. Let $G = (U, V, E)$ be a bipartile graph with $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$. The adjacency matrix A^G is a $n \times n$ matrix whose i, jth element is 1 if $(u_i, v_j) \in E$ and 0 otherwise. the determinant of A^G is $\det A^G = \sum_\pi \sigma(\pi) \Pi_{i=1}^n A^G_{i,\pi(i)}$.
3. G has a matching if and only if the determinant of $\det A^G$ is not identically zero.

PERMANENT

1. How many perfect matchings in a bipartile graph?
2. The permanent of A^G, $\text{perm } A^G = \sum_\pi \Pi_{i=1}^n A^G_{i,\pi(i)}$.
3. The number of perfect matchings in G is precisely the permanent of A^G.

Permanent

MATCHING

1. MATCHING: Whether a bipartile graph has a perfect matching?
2. Let $G = (U, V, E)$ be a bipartile graph with $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$. The adjacency matrix A^G is a $n \times n$ matrix whose i, jth element is 1 if $(u_i, v_j) \in E$ and 0 otherwise. the determinant of A^G is det $A^G = \sum_\pi \sigma(\pi) \Pi_{i=1}^n A^G_{i,\pi(i)}$.
3. G has a matching if and only if the determinant of det A^G is not identically zero.

PERMANENT

1. How many perfect matchings in a bipartile graph?
2. The permanent of A^G, perm $A^G = \sum_\pi \Pi_{i=1}^n A^G_{i,\pi(i)}$.
3. The number of perfect matchings in G is precisely the permanent of A^G.

Outline

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \le |x|^k$ for some $k \ge 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

The Class #P

Recall

1. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.
2. R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Definition

1. Let Q be a polynomially balanced and polynomial-time decidable binary relation. The **counting problem** associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$
2. #P is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.

Examples

1. Q is the relation "y satisfies expression x": #SAT;
2. "y is a Hamilton path of graph x": #HAMILTON PATH;
3. "y is a perfect matching in a bipartile graph x": PERMANENT.

#P-completeness

Definition

A **parsimonious reduction** from a counting problem A to a counting problem B is a function R which maps an instance x of A to an instance $R(x)$ of B such that the number of solutions of $R(x)$ is the same as that of x.

Definition

A counting problem in #P is **#P-complete** if every problem in #P can be reduced to it with a parsimonious reduction.

#P-completeness

Definition

A **parsimonious reduction** from a counting problem A to a counting problem B is a function R which maps an instance x of A to an instance $R(x)$ of B such that the number of solutions of $R(x)$ is the same as that of x.

Definition

A counting problem in #P is **#P-complete** if every problem in #P can be reduced to it with a parsimonious reduction.

#P-completeness(contd.)

Note

1. Most reductions between the decision problems in **NP** that we have seen there are indeed parsimonious reductions between the corresponding counting problems.
2. For example, CIRCUIT SAT to 3SAT. It is because the number of assignments where the output of circuit is true coincides with the number of satisfying assignments of the corresponding set of clauses.
3. But the reduction from 3SAT to HAMILTON PATH is not.

#P-completeness(contd.)

Note

1. Most reductions between the decision problems in **NP** that we have seen there are indeed parsimonious reductions between the corresponding counting problems.
2. For example, CIRCUIT SAT to 3SAT. It is because the number of assignments where the output of circuit is true coincides with the number of satisfying assignments of the corresponding set of clauses.
3. But the reduction from 3SAT to HAMILTON PATH is not.

#P-completeness(contd.)

Note

1. Most reductions between the decision problems in **NP** that we have seen there are indeed parsimonious reductions between the corresponding counting problems.
2. For example, CIRCUIT SAT to 3SAT. It is because the number of assignments where the output of circuit is true coincides with the number of satisfying assignments of the corresponding set of clauses.
3. But the reduction from 3SAT to HAMILTON PATH is not.

#P-completeness(contd..)

Theorem

#SAT is #P-complete.

Proof.

Given a problem $B \in$ #P with relation Q, we'll show it can be reduced to #CIRCUIT SAT with a parsimonious reduction, and hence can be reduced to 3SAT with a parsimonious reduction.

From the definition, there is a polynomial-time TM M deciding Q. We can build a circuit $C(x)$ with $|x|^k$ inputs such that with input y the output of $C(x)$ is true if and only if M accepts $x; y$ (Cook's Theorem). This is just a parsimonious reduction from B to #CIRCUIT SAT. □

#P-completeness(contd..)

Theorem

#SAT is #P-complete.

Proof.

Given a problem $B \in$ #P with relation Q, we'll show it can be reduced to #CIRCUIT SAT with a parsimonious reduction, and hence can be reduced to 3SAT with a parsimonious reduction.

From the definition, there is a polynomial-time TM M deciding Q. We can build a circuit $C(x)$ with $|x|^k$ inputs such that with input y the output of $C(x)$ is true if and only if M accepts $x; y$ (Cook's Theorem). This is just a parsimonious reduction from B to #CIRCUIT SAT. □

#P-completeness(contd..)

Theorem

#SAT is #P-complete.

Proof.

Given a problem $B \in$ #P with relation Q, we'll show it can be reduced to #CIRCUIT SAT with a parsimonious reduction, and hence can be reduced to 3SAT with a parsimonious reduction.
From the definition, there is a polynomial-time TM M deciding Q. We can build a circuit $C(x)$ with $|x|^k$ inputs such that with input y the output of $C(x)$ is true if and only if M accepts $x; y$ (Cook's Theorem). This is just a parsimonious reduction from B to #CIRCUIT SAT. $\square$

#P-completeness(contd..)

Theorem

PERMANENT is #P-complete.

Outline of proof

1. Reduction from #3SAT to WEIGHTED CYCLE COVERING (PERMANENT under integers).
2. Reduction from WEIGHTED CYCLE COVERING to CYCLE COVERING (0/1 PERMANENT).

#P-completeness(contd..)

Theorem

PERMANENT is #P-complete.

Outline of proof

1. Reduction from #3SAT to WEIGHTED CYCLE COVERING (PERMANENT under integers).
2. Reduction from WEIGHTED CYCLE COVERING to CYCLE COVERING ($0/1$ PERMANENT).

#P-completeness(contd..)

Theorem

#HAMILTON PATH is #P-complete.

Outline of proof

The reduction of 3SAT to HAMILTON PATH in the NP-complete proof is not a parsimonious, But there is a parsimonious reduction from 3SAT to HAMILTON PATH based on the reduction in Theorem 17.5 showing that TSP is FP^{NP}-complete.

#P-completeness(contd..)

Theorem

#HAMILTON PATH is #P-complete.

Outline of proof

The reduction of 3SAT to HAMILTON PATH in the NP-complete proof is not a parsimonious, But there is a parsimonious reduction from 3SAT to HAMILTON PATH based on the reduction in Theorem 17.5 showing that TSP is FP^{NP}-complete.

The relation between #P and some other classes

Note

1. #P problems can be solved in polynomial space.
2. Counting is stronger than the polynomial hierarchy.
3. (Toda's Theorem) $PH \subseteq P^{PP}$. It means polynomial oracle machines with a **PP** oracle can decide all languages in the polynomial hierarchy.
4. PP tells only whether the first bit of the number of accepting computations is 0 or 1; but #P asks for all n bits of this number.

The relation between #P and some other classes

Note

1. #P problems can be solved in polynomial space.
2. Counting is stronger than the polynomial hierarchy.
3. (Toda's Theorem) $PH \subseteq P^{PP}$. It means polynomial oracle machines with a **PP** oracle can decide all languages in the polynomial hierarchy.
4. PP tells only whether the first bit of the number of accepting computations is 0 or 1; but #P asks for all n bits of this number.

The relation between #P and some other classes

Note

1. #P problems can be solved in polynomial space.
2. Counting is stronger than the polynomial hierarchy.
3. (Toda's Theorem) $PH \subseteq P^{PP}$. It means polynomial oracle machines with a **PP** oracle can decide all languages in the polynomial hierarchy.
4. PP tells only whether the first bit of the number of accepting computations is 0 or 1; but #P asks for all n bits of this number.

Outline

Introduction

Introduction

1. $\oplus$SAT: given a set of clauses, is the number of satisfying assignments odd?
2. $\oplus$HAMILTON PATH: Given a graph, does it have an odd number of Hamilton paths?

Definition

A language L is said in the class $\oplus$P if there is a nondeterministic Turing Machine N such that for all string x, $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Equivalent definition

A language $L \in \oplus$P if there is a polynomially balanced and polynomially decidable relation R such that $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Introduction

Introduction

1. ⊕SAT: given a set of clauses, is the number of satisfying assignments odd?
2. ⊕HAMILTON PATH: Given a graph, does it have an odd number of Hamilton paths?

Definition

A language L is said in the class ⊕P if there is a nondeterministic Turing Machine N such that for all string x, $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Equivalent definition

A language $L \in \oplus$P if there is a polynomially balanced and polynomially decidable relation R such that $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Introduction

Introduction

1. $\oplus$SAT: given a set of clauses, is the number of satisfying assignments odd?
2. $\oplus$HAMILTON PATH: Given a graph, does it have an odd number of Hamilton paths?

Definition

A language L is said in the class $\oplus$P if there is a nondeterministic Turing Machine N such that for all string x, $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Equivalent definition

A language $L \in \oplus$P if there is a polynomially balanced and polynomially decidable relation R such that $x \in L$ if and only if the number of y's such that $(x, y) \in R$ is odd.

Outline

⊕P-completeness

Theorem

⊕SAT and ⊕HAMILTON PATH are ⊕P-complete.

Proof.

1. ⊕P can be easily seen from the equivalent definition.
2. ⊕P-completeness follows from the parsimonious reductions of any problem in #P to #SAT and #HAMILTON PATH.

□

$\oplus$P-completeness

Theorem

$\oplus$SAT and $\oplus$HAMILTON PATH are $\oplus$P-complete.

Proof.

1. $\oplus$P can be easily seen from the equivalent definition.
2. $\oplus$P-completeness follows from the parsimonious reductions of any problem in #P to #SAT and #HAMILTON PATH.

□

$\oplus$P-completeness

Theorem

$\oplus$SAT and $\oplus$HAMILTON PATH are $\oplus$P-complete.

Proof.

1. $\oplus$P can be easily seen from the equivalent definition.
2. $\oplus$P-completeness follows from the parsimonious reductions of any problem in #P to #SAT and #HAMILTON PATH.

□

⊕P-completeness (contd.)

Theorem

⊕P is closed under complement.

Proof.

The complement of ⊕SAT (deciding whether the number of satisfying assignments is even) is co⊕P-complete. We'll show that this problem reduces to ⊕SAT making ⊕SAT co⊕P-complete. Since ⊕SAT is also ⊕P-complete, ⊕P=co⊕P.

Now we need to show that the complement of ⊕SAT can be reduced to ⊕SAT: Given a set of clauses on variables $x_1, x_2, \ldots, x_n$.

(i) Add a new variable z to each clause;

(ii) add n clauses $\neg z \vee x_i$, $i = 1, 2, \ldots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.

(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

□

⊕P-completeness (contd.)

Theorem

⊕P is closed under complement.

Proof.

The complement of ⊕SAT (deciding whether the number of satisfying assignments is even) is co⊕P-complete. We'll show that this problem reduces to ⊕SAT making ⊕SAT co⊕P-complete. Since ⊕SAT is also ⊕P-complete,⊕P=co⊕P.

Now we need to show that the complement of ⊕SAT can be reduced to ⊕SAT: Given a set of clauses on variables $x_1, x_2, \ldots, x_n$.

(i) Add a new variable z to each clause;

(ii) add n clauses $\neg z \vee x_i$, $i = 1, 2, \ldots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.

(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

⊕P-completeness (contd.)

Theorem

⊕P is closed under complement.

Proof.

The complement of ⊕SAT (deciding whether the number of satisfying assignments is even) is co⊕P-complete. We'll show that this problem reduces to ⊕SAT making ⊕SAT co⊕P-complete. Since ⊕SAT is also ⊕P-complete,⊕P=co⊕P.
Now we need to show that the complement of ⊕SAT can be reduced to ⊕SAT: Given a set of clauses on variables $x_1, x_2, \ldots, x_n$.

(i) Add a new variable z to each clause;

(ii) add n clauses $\neg z \vee x_i$, $i = 1, 2, \ldots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.

(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

$\oplus$P-completeness (contd.)

Theorem

$\oplus$P is closed under complement.

Proof.

The complement of $\oplus$SAT (deciding whether the number of satisfying assignments is even) is co$\oplus$P-complete. We'll show that this problem reduces to $\oplus$SAT making $\oplus$SAT co$\oplus$P-complete. Since $\oplus$SAT is also $\oplus$P-complete,$\oplus$P=co$\oplus$P.

Now we need to show that the complement of $\oplus$SAT can be reduced to $\oplus$SAT: Given a set of clauses on variables $x_1, x_2, \ldots, x_n$.

(i) Add a new variable z to each clause;

(ii) add n clauses $\neg z \vee x_i$, $i = 1, 2, \ldots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.

(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

□

⊕P-completeness (contd.)

Theorem

⊕P is closed under complement.

Proof.

The complement of ⊕SAT (deciding whether the number of satisfying assignments is even) is co⊕P-complete. We'll show that this problem reduces to ⊕SAT making ⊕SAT co⊕P-complete. Since ⊕SAT is also ⊕P-complete,⊕P=co⊕P.
Now we need to show that the complement of ⊕SAT can be reduced to ⊕SAT: Given a set of clauses on variables $x_1, x_2, \ldots, x_n$.

(i) Add a new variable z to each clause;
(ii) add n clauses $\neg z \vee x_i, i = 1, 2, \ldots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.
(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

⊕P-completeness (contd.)

Theorem

⊕P is closed under complement.

Proof.

The complement of ⊕SAT (deciding whether the number of satisfying assignments is even) is co⊕P-complete. We'll show that this problem reduces to ⊕SAT making ⊕SAT co⊕P-complete. Since ⊕SAT is also ⊕P-complete,⊕P=co⊕P.
Now we need to show that the complement of ⊕SAT can be reduced to ⊕SAT: Given a set of clauses on variables $x_1, x_2, \dots, x_n$.

(i) Add a new variable z to each clause;
(ii) add n clauses $\neg z \vee x_i, i = 1, 2, \dots, n$.

The number of satisfying assignments will be increased by 1, because

(a) If $z =$ **false**, the satisfying assignment for the new clauses is one-to-one corresponding to the satisfying assignment for the old clauses.
(b) If $z =$ **true**, then all the other literals must be assigned **true**, in order to get a satisfying assignment.

□

Relation between $\oplus$P and some other classes

Note

1. $\oplus$P seems to be weaker than PP.
2. If an RP machine is equipped with an $\oplus$P oracle, it can simulate all of NP.

⊕P

Theorem

$\mathbf{NP} \subseteq \mathbf{RP}^{\oplus \mathbf{P}}$

Proof Sketch

1. The idea is to show how an NP-complete problem(SAT) can be solved using a Monte Carlo algorithm which uses ⊕SAT as its oracle.
2. Suppose we are dealing with a Boolean expression ϕ in CNF with n variables $x_1, x_2, \ldots, x_n$. Let S be a subset of $\{x_1, x_2, \ldots, x_n\}$. We define a Boolean expression η_S stating that an even number among the variables in S are **true**.
3. The basic idea is that if we continue to add the requirement that an even number of variables are **true** in a random subset, then with a reasonable probability one of the resulting expression has a single satisfying assignment, and thus its satisfiability can be detected by the ⊕SAT oracle.
4. Now an Monte Carlo algorithm for SAT using ⊕SAT as its oracle works as follows: Let ϕ_0 be the given expression ϕ. for $i = 1, 2, \ldots, n+1$, repeat the following:

⊕P

Theorem

NP $\subseteq$ **RP**$^{\oplus\mathbf{P}}$

Proof Sketch

1. The idea is to show how an NP-complete problem(SAT) can be solved using a Monte Carlo algorithm which uses ⊕SAT as its oracle.
2. Suppose we are dealing with a Boolean expression ϕ in CNF with n variables $x_1, x_2, \ldots, x_n$. Let S be a subset of $\{x_1, x_2, \ldots, x_n\}$. We define a Boolean expression η_S stating that an even number among the variables in S are **true**.
3. The basic idea is that if we continue to add the requirement that an even number of variables are **true** in a random subset, then with a reasonable probability one of the resulting expression has a single satisfying assignment, and thus its satisfiability can be detected by the ⊕SAT oracle.
4. Now an Monte Carlo algorithm for SAT using ⊕SAT as its oracle works as follows: Let ϕ_0 be the given expression ϕ. for $i = 1, 2, \ldots, n+1$, repeat the following:

⊕P

Theorem

NP ⊆ RP$^{\oplus\mathbf{P}}$

Proof Sketch

1. The idea is to show how an NP-complete problem(SAT) can be solved using a Monte Carlo algorithm which uses ⊕SAT as its oracle.
2. Suppose we are dealing with a Boolean expression ϕ in CNF with n variables $x_1, x_2, \ldots, x_n$. Let S be a subset of $\{x_1, x_2, \ldots, x_n\}$. We define a Boolean expression η_S stating that an even number among the variables in S are **true**.
3. The basic idea is that if we continue to add the requirement that an even number of variables are **true** in a random subset, then with a reasonable probability one of the resulting expression has a single satisfying assignment, and thus its satisfiability can be detected by the ⊕SAT oracle.
4. Now an Monte Carlo algorithm for SAT using ⊕SAT as its oracle works as follows: Let ϕ_0 be the given expression ϕ. for $i = 1, 2, \ldots, n+1$, repeat the following:

$\oplus$P

Theorem

$\mathbf{NP} \subseteq \mathbf{RP}^{\oplus\mathbf{P}}$

Proof Sketch

1. The idea is to show how an NP-complete problem(SAT) can be solved using a Monte Carlo algorithm which uses $\oplus$SAT as its oracle.
2. Suppose we are dealing with a Boolean expression ϕ in CNF with n variables $x_1, x_2, \ldots, x_n$. Let S be a subset of $\{x_1, x_2, \ldots, x_n\}$. We define a Boolean expression η_S stating that an even number among the variables in S are **true**.
3. The basic idea is that if we continue to add the requirement that an even number of variables are **true** in a random subset, then with a reasonable probability one of the resulting expression has a single satisfying assignment, and thus its satisfiability can be detected by the $\oplus$SAT oracle.
4. Now an Monte Carlo algorithm for SAT using $\oplus$SAT as its oracle works as follows: Let ϕ_0 be the given expression ϕ. for $i = 1, 2, \ldots, n+1$, repeat the following:

⊕P

Proof Sketch (contd.)

(i) Generate a random subset S_i of the variables and set $\phi_i = \phi_{i-1} \wedge \eta_{S_i}$.

(ii) If $\phi_i \in \oplus$SAT, then answer "ϕ is satisfiable";

(iii) If after $n + 1$ steps none of the ϕ_i's is in $\oplus$SAT, then answer "ϕ is probably unsatisfiable".