Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)
Υπολογιστική Πολυπλοκότητα

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών
Εθνικό Μετσόβιο Πολυτεχνείο

2019-2020

Πληροφορίες Μαθήματος

Θεωρητική Πληροφορική I (ΣΗΜΜΥ)
Υπολογιστική Πολυπλοκότητα (ΑΛΜΑ)

- Διδάσκοντες: Σ. Ζάχος, Ά. Παγουρτζής
- Βοηθοί Διδασκαλίας: Α. Αντωνόπουλος, Α. Χαλκή
- Επιμέλεια Διαφανειών: Α. Αντωνόπουλος
- Δευτέρα: 17:00 - 20:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
  Πέμπτη: 15:00 - 17:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
- Ώρες Γραφείου: Μετά από κάθε μάθημα
- Σελίδα: http://courses.corelab.ntua.gr/complexity
- Βαθμολόγηση:

Διαγώνισμα:	6 μονάδες
Ασκήσεις:	2 μονάδες
Ομιλία:	2 μονάδες
Quiz:	1 μονάδα

Computational Complexity

Graduate Course

Antonis Antonopoulos

Computation and Reasoning Laboratory
National Technical University of Athens

Bibliography

Textbooks

1. C. Papadimitriou, **Computational Complexity**, Addison Wesley, 1994
2. S. Arora, B. Barak, **Computational Complexity: A Modern Approach**, Cambridge University Press, 2009
3. O. Goldreich, **Computational Complexity: A Conceptual Perspective**, Cambridge University Press, 2008

Lecture Notes

1. L. Trevisan, **Lecture Notes in Computational Complexity**, 2002, UC Berkeley
2. J. Katz, **Notes on Complexity Theory**, 2011, University of Maryland
3. Jin-Yi Cai, **Lectures in Computational Complexity**, 2003, University of Wisconsin Madison

Contents

- **Computational Complexity**: Quantifying the amount of computational resources required to solve a given task. *Classify* computational problems according to their inherent difficulty in complexity classes, and prove relations among them.

- **Structural Complexity**: “The study of the relations between various complexity classes and the global properties of individual classes. [...] The goal of structural complexity is a *thorough understanding of the relations between the various complexity classes and the internal structure of these complexity classes*.” [J. Hartmanis]

Decision Problems

- Have answers of the form “*yes*” or “*no*”.
- Encoding: each instance x of the problem is represented as a *string* of an alphabet Σ ($|\Sigma| \geq 2$).
- Decision problems have the form “Is x in L?”, where L is a *language*, $L \subseteq \Sigma^*$.

- So, for an encoding of the input, using the alphabet Σ, we associate the following language with the decision problem Π:

$$L(\Pi) = \{x \in \Sigma^* \mid x \text{ is a representation of a “yes” instance of the problem } \Pi\}$$

Example

- Given a number x, is this number prime? ($x \overset{?}{\in}$ PRIMES)
- Given graph G and a number k, is there a clique with k (or more) nodes in G?

Search Problems

- Have answers of the form of an **object**.
- **Relation** $R(x, y)$ connecting instances x with answers (objects) y we wish to find for x.
- Given instance x, find a y such that $(x, y) \in R$.

Search Problems

- Have answers of the form of an **object**.
- **Relation** $R(x, y)$ connecting instances x with answers (objects) y we wish to find for x.
- Given instance x, find a y such that $(x, y) \in R$.

Example

FACTORING: Given integer N, find its prime decomposition:

$$N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$$

Optimization Problems

- For each instance x there is a **set of Feasible Solutions** $F(x)$.
- To each $s \in F(x)$ we map a positive integer $c(x)$, using **the objective function** $c(s)$.
- We search for the solution $s \in F(x)$ which minimizes (or maximizes) the objective function $c(s)$.

Optimization Problems

- For each instance x there is a **set of Feasible Solutions $F(x)$**.
- To each $s \in F(x)$ we map a positive integer $c(x)$, using **the objective function $c(s)$**.
- We search for the solution $s \in F(x)$ which minimizes (or maximizes) the objective function $c(s)$.

Example

- The **Traveling Salesperson Problem** (TSP):
  Given a finite set $C = \{c_1, \ldots, c_n\}$ of cities and a distance $d(c_i, c_j) \in \mathbb{Z}^+, \forall (c_i, c_j) \in C^2$, we ask for a permutation π of C, that minimizes this quantity:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

A Model Discussion

- There are many computational models (RAM, Turing Machines etc).
- The **Church-Turing Thesis** states that all computation models are equivalent. That is, every computation model can be simulated by a Turing Machine.

A Model Discussion

- There are many computational models (RAM, Turing Machines etc).
- The **Church-Turing Thesis** states that all computation models are equivalent. That is, every computation model can be simulated by a Turing Machine.
- In Complexity Theory, we consider **efficiently computable** the problems which are solved (aka the languages that are decided) in **polynomial number of steps** (*Edmonds-Cobham Thesis*).

A Model Discussion

- There are many computational models (RAM, Turing Machines etc).
- The **Church-Turing Thesis** states that all computation models are equivalent. That is, every computation model can be simulated by a Turing Machine.
- In Complexity Theory, we consider **efficiently computable** the problems which are solved (aka the languages that are decided) in **polynomial number of steps** (*Edmonds-Cobham Thesis*).

$$\boxed{\textbf{Efficiently Computable} \equiv \textbf{Polynomial-Time Computable}}$$

Summary

- Computational Complexity classifies problems into classes, and studies the relations and the structure of these classes.
- We have decision problems with boolean answer, or function/optimization problems which output an object as an answer.
- Given some nice properties of polynomials, we identify polynomial-time algorithms as efficient algorithms.

Contents

Definition

A Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{S, L, R\}$ is the transition function.

- A TM is a “programming language” with a single data structure (a tape), and a cursor, which moves left and right on the tape.
- Function δ is the **program** of the machine.

Turing Machines and Languages

Definition

Let $L \subseteq \Sigma^*$ be a language and M a TM such that, for every string $x \in \Sigma^*$:

- If $x \in L$, then $M(x) =$ "yes"
- If $x \notin L$, then $M(x) =$ "no"

Then we say that ***M* decides *L***.

- Alternatively, we say that $M(x) = L(x)$, where $L(x) = \chi_L(x)$ is the *characteristic function* of L (if we consider 1 as "yes" and 0 as "no").
- If L is decided by some TM M, then L is called a **recursive language**.

Definition

If for a language L there is a TM M, which if $x \in L$ then $M(x) =$ "yes", and if $x \notin L$ then $M(x) \uparrow$, we call L **recursively enumerable**.

*By $M(x) \uparrow$ we mean that M does not halt on input x (it runs forever).

Theorem

If L is recursive, then it is recursively enumerable.

Proof: *Exercise*

Definition

If for a language L there is a TM M, which if $x \in L$ then $M(x) =$ "yes", and if $x \notin L$ then $M(x) \uparrow$, we call L **recursively enumerable**.

*By $M(x) \uparrow$ we mean that M does not halt on input x (it runs forever).

Theorem

If L is recursive, then it is recursively enumerable.

Proof: *Exercise*

Definition

If f is a function, $f : \Sigma^* \to \Sigma^*$, we say that a TM M computes f if, for any string $x \in \Sigma^*$, $M(x) = f(x)$. If such M exists, f is called a **recursive function**.

- Turing Machines can be thought as algorithms for solving string related problems.

Multitape Turing Machines

- We can extend the previous Turing Machine definition to obtain a Turing Machine with multiple tapes:

Definition

A k-tape Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta : (Q \setminus F) \times \Gamma^k \to Q \times (\Gamma \times \{S, L, R\})^k$ is the transition function.

Bounds on Turing Machines

- We will characterize the "performance" of a Turing Machine by the amount of *time* and *space* required on instances of size n, when these amounts are expressed as a function of n.

Definition

Let $T : \mathbb{N} \rightarrow \mathbb{N}$. We say that machine M operates within time $T(n)$ if, for any input string x, the time required by M to reach a final state is at most $T(|x|)$. Function T is a **time bound** for M.

Definition

Let $S : \mathbb{N} \rightarrow \mathbb{N}$. We say that machine M operates within space $S(n)$ if, for any input string x, M visits at most $S(|x|)$ locations on its work tapes (excluding the input tape) during its computation. Function S is a **space bound** for M.

Multitape Turing Machines

Theorem

Given any k-tape Turing Machine M operating within time $T(n)$, we can construct a TM M' operating within time $\mathcal{O}\left(T^2(n)\right)$ such that, for any input $x \in \Sigma^*$, $M(x) = M'(x)$.

Proof: See Th.2.1 (p.30) in [1].

Multitape Turing Machines

Theorem

Given any k-tape Turing Machine M operating within time $T(n)$, we can construct a TM M' operating within time $\mathcal{O}\left(T^2(n)\right)$ such that, for any input $x \in \Sigma^*$, $M(x) = M'(x)$.

Proof: See Th.2.1 (p.30) in [1].

This is a strong evidence of the robustness of our model:
Adding a bounded number of strings does not increase their computational capabilities, and affects their efficiency only polynomially.

Linear Speedup

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within time $T(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within time $T'(n) = \varepsilon T(n) + n + 2$.

Proof: See Th.2.2 (p.32) in [1].

Linear Speedup

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within time $T(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within time $T'(n) = \varepsilon T(n) + n + 2$.

Proof: See Th.2.2 (p.32) in [1].

- If, for example, T is linear, i.e. something like cn, then this theorem states that the constant c can be made arbitrarily close to 1. So, it is fair to start using the $\mathcal{O}(\cdot)$ notation in our time bounds.
- A similar theorem holds for space:

Linear Speedup

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within time $T(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within time $T'(n) = \varepsilon T(n) + n + 2$.

Proof: See Th.2.2 (p.32) in [1].

- If, for example, T is linear, i.e. something like cn, then this theorem states that the constant c can be made arbitrarily close to 1. So, it is fair to start using the $\mathcal{O}(\cdot)$ notation in our time bounds.
- A similar theorem holds for space:

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within space $S(n)$. Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within space $S'(n) = \varepsilon S(n) + 2$.

Nondeterministic Turing Machines

- We will now introduce an **unrealistic** model of computation:

Definition

A Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta : (Q \setminus F) \times \Gamma \to Pow(Q \times \Gamma \times \{S, L, R\})$ is the transition **relation**.

Nondeterministic Turing Machines

- In this model, an input is accepted if **there is** *some sequence* of nondeterministic choices that results in “yes”.
- An input is rejected if there is *no sequence* of choices that lead to acceptance.
- Observe the similarity with recursively enumerable languages.

Nondeterministic Turing Machines

- In this model, an input is accepted if **there is** *some sequence* of nondeterministic choices that results in “yes”.
- An input is rejected if there is *no sequence* of choices that lead to acceptance.
- Observe the similarity with recursively enumerable languages.

Definition

We say that M operates within bound $T(n)$, if for every input $x \in \Sigma^*$ and every sequence of nondeterministic choices, M reaches a final state within $T(|x|)$ steps.

Nondeterministic Turing Machines

- In this model, an input is accepted if **there is** *some sequence* of nondeterministic choices that results in “yes”.
- An input is rejected if there is *no sequence* of choices that lead to acceptance.
- Observe the similarity with recursively enumerable languages.

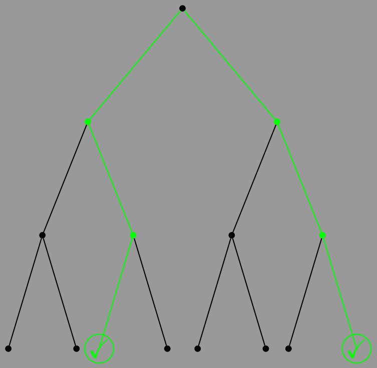
Definition

We say that M operates within bound $T(n)$, if for every input $x \in \Sigma^*$ and every sequence of nondeterministic choices, M reaches a final state within $T(|x|)$ steps.

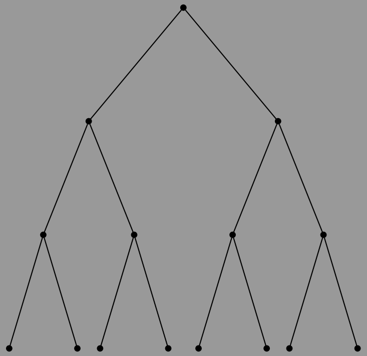
- The above definition requires that M does not have computation paths longer than $T(n)$, where $n = |x|$ the length of the input.
- The amount of time charged is the *depth* of the **computation tree**.

Examples of Nondeterministic Computations

Example



Accepting computation



Rejecting Computation

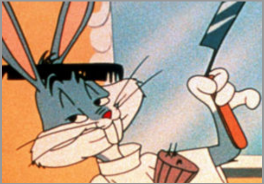
- Without loss of generality, the computation trees are binary, full and complete. (*why*?)

Summary

- A recursive language is decided by a TM.
- A recursive enumerable language is accepted by a TM that halts only if $x \in L$.
- Multiple tape TMs can be simulated by a one-tape TM with quadratic overhead.
- Linear speedup justifies the $\mathcal{O}(\cdot)$ notation.
- Nondeterministic TMs move in “parallel universes”, making different choices simultaneously.
- A Deterministic TM computation is a *path*.
- A Nondeterministic TM computation is a *tree*, i.e. exponentially many paths ran simultaneously.

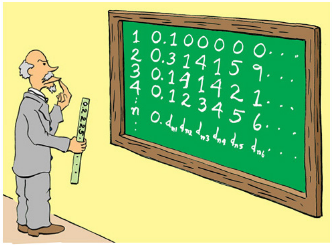
Contents

Diagonalization



Suppose there is a town with just one barber, who is male. In this town, the barber shaves all those, and only those, men in town who do not shave themselves. Who shaves the barber?

Diagonalization is a technique that was used in many different cases:



George showed it wouldn't fit in.

Diagonalization

Theorem

The functions from $\mathbb{N}$ to $\mathbb{N}$ are uncountable.

Proof: Let, for the sake of contradiction that are countable: $\phi_1, \phi_2, \ldots$. Consider the following function: $f(x) = \phi_x(x) + 1$. This function must appear somewhere in this enumeration, so let $\phi_y = f(x)$. Then $\phi_y(x) = \phi_x(x) + 1$, and if we choose y as an argument, then $\phi_y(y) = \phi_y(y) + 1$. $\square$

Diagonalization

Theorem

The functions from $\mathbb{N}$ to $\mathbb{N}$ are uncountable.

Proof: Let, for the sake of contradiction that are countable: $\phi_1, \phi_2, \ldots$. Consider the following function: $f(x) = \phi_x(x) + 1$. This function must appear somewhere in this enumeration, so let $\phi_y = f(x)$. Then $\phi_y(x) = \phi_x(x) + 1$, and if we choose y as an argument, then $\phi_y(y) = \phi_y(y) + 1$. □

- Using the same argument:

Theorem

The functions from $\{0, 1\}^*$ to $\{0, 1\}$ are uncountable.

Machines as strings

- It is obvious that we can represent a Turing Machine as a string: *just write down the description and encode it using an alphabet, e.g.* $\{0,1\}$.
- We denote by $\llcorner M \lrcorner$ the TM M's representation as a string.
- Also, if $x \in \Sigma^*$, we denote by M_x the TM that x represents.

Keep in mind that:

- **Every string represents *some* TM.**
- **Every TM is represented by *infinitely many* strings.**

- There exists (*at least*) an uncomputable function from $\{0,1\}^*$ to $\{0,1\}$, since the set of all TMs is countable.

The Universal Turing Machine

- So far, our computational models are specified to solve a single problem.
- Turing observed that there is a TM that can simulate any other TM M, given M's description as input.

Theorem

There exists a TM $\mathcal{U}$ such that for every $x, w \in \Sigma^*$, $\mathcal{U}(x, w) = M_w(x)$. Also, if M_w halts within T steps on input x, then $\mathcal{U}(x, w)$ halts within $CT \log T$ steps, where C is a constant independent of x, and depending only on M_w's alphabet size number of tapes and number of states.

Proof: See section 3.1 in [1], and Th. 1.9 and section 1.7 in [2].

The Halting Problem

- Consider the following problem: "*Given the description of a TM M, and a string x, will M halt on input x?*" This is called the HALTING PROBLEM.
- **We want to compute this problem ! ! !** (Given a computer program and an input, will this program enter an infinite loop?)
- In language form: H $= \{\llcorner M \lrcorner; x \mid M(x)\downarrow\}$, where "$\downarrow$" means that the machine halts, and "$\uparrow$" that it runs forever.

Theorem

H *is recursively enumerable.*

Proof: See Th.3.1 (p.59) in [1]

- In fact, H is not just a recursively enumerable language:
  If we had an algorithm for deciding H, then we would be able to derive an algorithm for deciding any r.e. language (**RE**-complete).

The Halting Problem

- But....

Theorem

H *is not recursive.*

Proof: See Th.3.1 (p.60) in [1]

- Suppose, for the sake of contradiction, that there is a TM M_H that decides H.
- Consider the TM D:
$$\boxed{D(\llcorner M \lrcorner) : \textbf{ if } M_H(\llcorner M \lrcorner; \llcorner M \lrcorner) = \text{“yes” } \textbf{then} \uparrow \textbf{ else } \text{“yes”}}$$
- What is $D(\llcorner D \lrcorner)$?

The Halting Problem

- But....

Theorem

H *is not recursive.*

Proof: See Th.3.1 (p.60) in [1]

- Suppose, for the sake of contradiction, that there is a TM M_H that decides H.
- Consider the TM D:
  $\boxed{D(\llcorner M \lrcorner) : \textbf{ if } M_H(\llcorner M \lrcorner; \llcorner M \lrcorner) = \text{“yes” } \textbf{then } \uparrow \textbf{ else } \text{ “yes”}}$
- What is $D(\llcorner D \lrcorner)$?
- If $D(\llcorner D \lrcorner)\uparrow$, then M_H accepts the input, so $\llcorner D \lrcorner; \llcorner D \lrcorner \in$ H, so $D(D)\downarrow$.
- If $D(\llcorner D \lrcorner)\downarrow$, then M_H rejects $\llcorner D \lrcorner; \llcorner D \lrcorner$, so $\llcorner D \lrcorner; \llcorner D \lrcorner \notin$ H, so $D(D)\uparrow$. $\square$

- Recursive languages are a *proper* subset of recursive enumerable ones.
- Recall that the complement of a language L is defined as:

$$\overline{L} = \{x \in \Sigma^* \mid x \notin L\} = \Sigma^* \setminus L$$

Theorem

1. *If L is recursive, so is $\overline{L}$.*
2. *L is recursive if and only if L and $\overline{L}$ are recursively enumerable.*

Proof: Exercise

- Recursive languages are a *proper* subset of recursive enumerable ones.
- Recall that the complement of a language L is defined as:

$$\overline{L} = \{x \in \Sigma^* \mid x \notin L\} = \Sigma^* \setminus L$$

Theorem

1. *If L is recursive, so is $\overline{L}$.*
2. *L is recursive if and only if L and $\overline{L}$ are recursively enumerable.*

Proof: Exercise

- Let $E(M) = \{x \mid (q_0, \triangleright, \varepsilon) \overset{M*}{\rightarrow} (q, y \sqcup x \sqcup, \varepsilon\}$
- $E(M)$ is the language *enumerated* by M.

Theorem

L is recursively enumerable iff there is a TM M such that $L = E(M)$.

More Undecidability

- The HALTING PROBLEM, our first undecidable problem, was the first, but not the only undecidable problem. Its spawns a wide range of such problems, via *reductions*.
- To show that a problem A is undecidable we establish that, if there is an algorithm for A, then there would be an algorithm for H, which is absurd.

More Undecidability

- The HALTING PROBLEM, our first undecidable problem, was the first, but not the only undecidable problem. Its spawns a wide range of such problems, via *reductions*.
- To show that a problem A is undecidable we establish that, if there is an algorithm for A, then there would be an algorithm for H, which is absurd.

Theorem

The following languages are not recursive:

1. $\{\llcorner M \lrcorner \mid M \textit{ halts on all inputs}\}$
2. $\{\llcorner M \lrcorner ; x \mid \textit{There is a } y \textit{ such that } M(x) = y\}$
3. $\{\llcorner M \lrcorner ; x \mid \textit{The computation of } M \textit{ uses all states of } M\}$
4. $\{\llcorner M \lrcorner ; x ; y \mid M(x) = y\}$

Rice’s Theorem

- The previous problems lead us to a more general conclusion:

Any non-trivial property of Turing Machines is undecidable

- If a TM M accepts a language L, we write $L = L(M)$.

Theorem (Rice’s Theorem)

Suppose that $\mathcal{C}$ is a proper, non-empty subset of the set of all recursively enumerable languages. Then, the following problem is undecidable:

Given a Turing Machine M, is $L(M) \in \mathcal{C}$?

Rice's Theorem

Proof: See Th.3.2 (p.62) in [1]

- We can assume that $\emptyset \notin \mathcal{C}$ (*why?*).
- Since $\mathcal{C}$ is nonempty, $\exists\ L \in \mathcal{C}$, accepted by the TM M_L.
- Let M_H the TM accepting the HALTING PROBLEM for an arbitrary input x. For each $x \in \Sigma^*$, we construct a TM M as follows:

  $$\boxed{M(y) :\ \textbf{if}\ M_H(x) = \text{“yes”}\ \textbf{then}\ M_L(y)\ \textbf{else}\ \uparrow}$$
- We claim that: $L(M) \in \mathcal{C}$ if and only if $x \in \mathrm{H}$.

Rice's Theorem

Proof:

See Th.3.2 (p.62) in [1]

- We can assume that $\emptyset \notin \mathcal{C}$ (*why?*).
- Since $\mathcal{C}$ is nonempty, $\exists\ L \in \mathcal{C}$, accepted by the TM M_L.
- Let M_H the TM accepting the HALTING PROBLEM for an arbitrary input x. For each $x \in \Sigma^*$, we construct a TM M as follows:

  $$\boxed{M(y)\ :\ \textbf{if } M_H(x) = \text{“yes”} \textbf{ then } M_L(y) \textbf{ else } \uparrow}$$
- We claim that: $L(M) \in \mathcal{C}$ if and only if $x \in \mathrm{H}$.
  Proof of the claim:
  - If $x \in \mathrm{H}$, then $M_H(x) = \text{“yes”}$, and so M will accept y or never halt, depending on whether $y \in L$. Then the language accepted by M is exactly L, which is in $\mathcal{C}$.
  - If $M_H(x) \uparrow$, M never halts, and thus M accepts the language $\emptyset$, which is not in $\mathcal{C}$. $\square$

Summary

- TMs are encoded by strings.
- The Universal TM $\mathcal{U}(x, \llcorner M \lrcorner)$ can simulate any other TM M along with an input x.
- The Halting Problem is recursively enumerable, but not recursive.
- Many other problems can be proved undecidable, by a *reduction* from the Halting Problem.
- Rice's theorem states that *any non-trivial property of TMs is an undecidable problem*.

Contents

Parameters used to define complexity classes:

- Model of Computation (Turing Machine, RAM, Circuits)
- Mode of Computation (Deterministic, Nondeterministic, Probabilistic)
- Complexity Measures (*Time, Space, Circuit Size-Depth*)
- Other Parameters (Randomization, Interaction)

Our first complexity classes

Definition

Let $L \subseteq \Sigma^*$, and $T, S : \mathbb{N} \to \mathbb{N}$:

- We say that $L \in \mathbf{DTIME}[T(n)]$ if there exists a TM M deciding L, which operates within the *time* bound $\mathcal{O}\left(T(n)\right)$, where $n = |x|$.
- We say that $L \in \mathbf{DSPACE}[S(n)]$ if there exists a TM M deciding L, which operates within *space* bound $\mathcal{O}\left(S(n)\right)$, that is, for any input x, requires space at most $S(|x|)$.
- We say that $L \in \mathbf{NTIME}[T(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within the time bound $\mathcal{O}\left(T(n)\right)$.
- We say that $L \in \mathbf{NSPACE}[S(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within space bound $\mathcal{O}\left(S(n)\right)$.

Our first complexity classes

- The above are **Complexity Classes**, in the sense that they are sets of languages.
- All these classes are parameterized by a function T or S, so they are *families* of classes (for each function we obtain a complexity class).

Our first complexity classes

- The above are **Complexity Classes**, in the sense that they are sets of languages.
- All these classes are parameterized by a function T or S, so they are *families* of classes (for each function we obtain a complexity class).

Definition (Complementary complexity class)

For any complexity class $\mathcal{C}$, $co\mathcal{C}$ denotes the class: $\{\overline{L} \mid L \in \mathcal{C}\}$, where $\overline{L} = \Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$.

- We want to define "reasonable" complexity classes, in the sense that we want to "compute more problems", given more computational resources.

Constructible Functions

- Can we use all computable functions to define Complexity Classes?

Constructible Functions

- Can we use all computable functions to define Complexity Classes?

Theorem (Gap Theorem)

For any computable functions r and a, there exists a computable function f such that $f(n) \geq a(n)$, and

$$\mathbf{DTIME}[f(n)] = \mathbf{DTIME}[r(f(n))]$$

- That means, for $r(n) = 2^{2^{f(n)}}$, the incementation from $f(n)$ to $2^{2^{f(n)}}$ does not allow the computation of any new function!
- So, we must use some restricted families of functions:

Constructible Functions

Definition (Time-Constructible Function)

A nondecreasing function $T : \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$.

Definition (Space-Constructible Function)

A nondecreasing function $S : \mathbb{N} \to \mathbb{N}$ is **space-constructible** if $S(n) > \log n$ and there is a TM M that computes $S(|x|)$ using $S(|x|)$ space, given x as input.

Constructible Functions

Definition (Time-Constructible Function)

A nondecreasing function $T : \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$.

Definition (Space-Constructible Function)

A nondecreasing function $S : \mathbb{N} \to \mathbb{N}$ is **space-constructible** if $S(n) > \log n$ and there is a TM M that computes $S(|x|)$ using $S(|x|)$ space, given x as input.

- The restriction $T(n) \geq n$ is to allow the machine to read its input.
- The restriction $S(n) > \log n$ is to allow the machine to “remember” the index of the cell of the input tape that it is currently reading.

Constructible Functions

- Also, if $f_1(n), f_2(n)$ are time/space-constructible functions, so are $f_1 + f_2, f_1 \cdot f_2$ and $f_1^{f_2}$.

Constructible Functions

- Also, if $f_1(n), f_2(n)$ are time/space-constructible functions, so are $f_1 + f_2, f_1 \cdot f_2$ and $f_1^{f_2}$.
- If we use only *constructible* functions, we can prove **Hierarchy Theorems**, stating that with more resources we can compute more languages:

Constructible Functions

- Also, if $f_1(n), f_2(n)$ are time/space-constructible functions, so are $f_1 + f_2, f_1 \cdot f_2$ and $f_1^{f_2}$.
- If we use only *constructible* functions, we can prove **Hierarchy Theorems**, stating that with more resources we can compute more languages:

Theorem (Hierarchy Theorems)

Let t_1, t_2 be time-constructible functions, and s_1, s_2 be space-constructible functions. Then:

1. *If* $t_1(n) \log t_1(n) = o(t_2(n))$, *then* $\mathbf{DTIME}(t_1) \subsetneq \mathbf{DTIME}(t_2)$.
2. *If* $t_1(n+1) = o(t_2(n))$, *then* $\mathbf{NTIME}(t_1) \subsetneq \mathbf{NTIME}(t_2)$.
3. *If* $s_1(n) = o(s_2(n))$, *then* $\mathbf{DSPACE}(s_1) \subsetneq \mathbf{DSPACE}(s_2)$.
4. *If* $s_1(n) = o(s_2(n))$, *then* $\mathbf{NSPACE}(s_1) \subsetneq \mathbf{NSPACE}(s_2)$.

Simplified Case of Deterministic Time Hierarchy Theorem

Theorem

$\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^{1.5}]$

Simplified Case of Deterministic Time Hierarchy Theorem

Theorem

$\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^{1.5}]$

Proof (*Diagonalization*):

See Th.3.1 (p.69) in [2]

Let D be the following machine:

```
On input $x$, run for $|x|^{1.4}$ steps $\mathcal{U}(M_x, x)$;
If $\mathcal{U}(M_x, x) = b$, then return $1 - b$;
Else return 0;
```

- Clearly, $L = L(D) \in \mathbf{DTIME}[n^{1.5}]$
- We claim that $L \notin \mathbf{DTIME}[n]$:
  Let $L \in \mathbf{DTIME}[n] \Rightarrow \exists\, M: \ M(x) = D(x)\ \forall x \in \Sigma^*$, and M works for $\mathcal{O}(|x|)$ steps.
  The time to simulate M using $\mathcal{U}$ is $c|x|\log|x|$, for some c.

Simplified Case of Deterministic Time Hierarchy Theorem

Proof (*cont'd*):
$\exists n_0 : \; n^{1.4} > cn \log n \; \forall n \geq n_0$
There exists a x_M, s.t. $x_M = \llcorner M \lrcorner$ and $|x_M| > n_0$ (*why?*) Then,
$D(x_M) = 1 - M(x_M)$ (while we have also that $D(x) = M(x), \; \forall x$)

Simplified Case of Deterministic Time Hierarchy Theorem

Proof (*cont'd*):
$\exists n_0 : n^{1.4} > cn \log n \; \forall n \geq n_0$
There exists a x_M, s.t. $x_M = \llcorner M \lrcorner$ and $|x_M| > n_0$ (*why?*) Then,
$\mathbf{D(x_M) = 1 - M(x_M)}$ (while we have also that $D(x) = M(x), \; \forall x$)
Contradiction!! $\square$

Simplified Case of Deterministic Time Hierarchy Theorem

Proof (*cont'd*):
$\exists n_0 : \; n^{1.4} > cn \log n \; \forall n \geq n_0$
There exists a x_M, s.t. $x_M = \llcorner M \lrcorner$ and $|x_M| > n_0$ (*why?*) Then,
$\mathbf{D(x_M) = 1 - M(x_M)}$ (while we have also that $D(x) = M(x), \; \forall x$)
Contradiction!! $\square$

- So, we have the hierachy:

$$\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^2] \subsetneq \mathbf{DTIME}[n^3] \subsetneq \cdots$$

- We will later see that the class containing the problems we can efficiently solve (recall the Edmonds-Cobham Thesis) is the class $\mathbf{P} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[n^c]$.

- Hierarchy Theorems tell us how classes of the same kind relate to each other, when we vary the complexity bound.
- The most interesting results concern relationships between classes of different kinds:

- Hierarchy Theorems tell us how classes of the same kind relate to each other, when we vary the complexity bound.
- The most interesting results concern relationships between classes of different kinds:

Theorem

Suppose that $T(n), S(n)$ are time-constructible and space-constructible functions, respectively.Then:

1. $\mathbf{DTIME}[T(n)] \subseteq \mathbf{NTIME}[T(n)]$
2. $\mathbf{DSPACE}[S(n)] \subseteq \mathbf{NSPACE}[S(n)]$
3. $\mathbf{NTIME}[T(n)] \subseteq \mathbf{DSPACE}[T(n)]$
4. $\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DTIME}[2^{\mathcal{O}(S(n))}]$

- Hierarchy Theorems tell us how classes of the same kind relate to each other, when we vary the complexity bound.
- The most interesting results concern relationships between classes of different kinds:

Theorem

Suppose that $T(n), S(n)$ are time-constructible and space-constructible functions, respectively.Then:

1. $\mathbf{DTIME}[T(n)] \subseteq \mathbf{NTIME}[T(n)]$
2. $\mathbf{DSPACE}[S(n)] \subseteq \mathbf{NSPACE}[S(n)]$
3. $\mathbf{NTIME}[T(n)] \subseteq \mathbf{DSPACE}[T(n)]$
4. $\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DTIME}[2^{\mathcal{O}(S(n))}]$

Corollary

$$\mathbf{NTIME}[T(n)] \subseteq \bigcup_{c>1} \mathbf{DTIME}[c^{T(n)}]$$

Proof: See Th.7.4 (p.147) in [1]

1. Trivial
2. Trivial
3. We can simulate the machine for each nondeterministic choice, using at most $T(n)$ steps in each simulation.
   There are *exponentially* many simulations, but we can simulate them one-by-one, *reusing the same space*.
4. Recall the notion of a configuration of a TM: For a k-tape machine, is a $2k-2$ tuple: $(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1})$
   How many configurations are there?
   - $|Q|$ choices for the state
   - $n+1$ choices for i, and
   - Fewer than $|\Sigma|^{(2k-2)S(n)}$ for the remaining strings

   So, the total number of configurations on input size n is at most $nc_1^{S(n)} = 2^{\mathcal{O}(S(n))}$.

Proof (*cont'd*):

Definition (Configuration Graph of a TM)

The configuration graph of M on input x, denoted $G(M,x)$, has as **vertices** all the possible configurations, and there is an **edge** between two vertices C and C' if and only if C' can be reached from C in one step, according to M's transition function.

Proof (*cont'd*):

Definition (Configuration Graph of a TM)

The configuration graph of M on input x, denoted $G(M,x)$, has as **vertices** all the possible configurations, and there is an **edge** between two vertices C and C' if and only if C' can be reached from C in one step, according to M's transition function.

- So, we have reduced this simulation to REACHABILITY* problem (also known as S-T CONN), for which we know there is a poly-time ($\mathcal{O}\left(n^2\right)$) algorithm.
- So, the simulation takes $\left(2^{\mathcal{O}(S(n))}\right)^2 \sim 2^{\mathcal{O}(S(n))}$ steps. □

*REACHABILITY: Given a graph G and two nodes $v_1, v_n \in V$, is there a path from v_1 to v_n?

The essential Complexity Hierarchy

Definition

$$\mathbf{L} = \mathbf{DSPACE}[\log n]$$

$$\mathbf{NL} = \mathbf{NSPACE}[\log n]$$

$$\mathbf{P} = \bigcup_{c\in\mathbb{N}} \mathbf{DTIME}[n^c]$$

$$\mathbf{NP} = \bigcup_{c\in\mathbb{N}} \mathbf{NTIME}[n^c]$$

$$\mathbf{PSPACE} = \bigcup_{c\in\mathbb{N}} \mathbf{DSPACE}[n^c]$$

$$\mathbf{NPSPACE} = \bigcup_{c\in\mathbb{N}} \mathbf{NSPACE}[n^c]$$

The essential Complexity Hierarchy

Definition

$$\mathbf{EXP} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[2^{n^c}]$$

$$\mathbf{NEXP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[2^{n^c}]$$

$$\mathbf{EXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[2^{n^c}]$$

$$\mathbf{NEXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[2^{n^c}]$$

The essential Complexity Hierarchy

Definition

$$\mathbf{EXP} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[2^{n^c}]$$

$$\mathbf{NEXP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[2^{n^c}]$$

$$\mathbf{EXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[2^{n^c}]$$

$$\mathbf{NEXPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[2^{n^c}]$$

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$$

Certificate Characterization of NP

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ a binary relation on strings.

- R is called **polynomially decidable** if there is a DTM deciding the language $\{x; y \mid (x, y) \in R\}$ in polynomial time.
- R is called **polynomially balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$, for some $k \geq 1$.

Certificate Characterization of NP

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ a binary relation on strings.

- R is called **polynomially decidable** if there is a DTM deciding the language $\{x; y \mid (x, y) \in R\}$ in polynomial time.
- R is called **polynomially balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$, for some $k \geq 1$.

Theorem

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomially decidable and polynomially balanced relation R, such that:*

$$L = \{x \mid \exists y \, R(x, y)\}$$

- This y is called **succinct certificate**, or **witness**.
- So, an **NP** **Search Problem** is the problem of *computing* witnesses.

Proof: See Pr.9.1 (p.181) in [1]

($\Leftarrow$) If such an R exists, we can construct the following NTM deciding L:
"On input x, *guess* a y, such that $|y| \leq |x|^k$, and then test (in poly-time) if $(x, y) \in R$. If so, accept, else reject." Observe that an accepting computation exists if and only if $x \in L$.

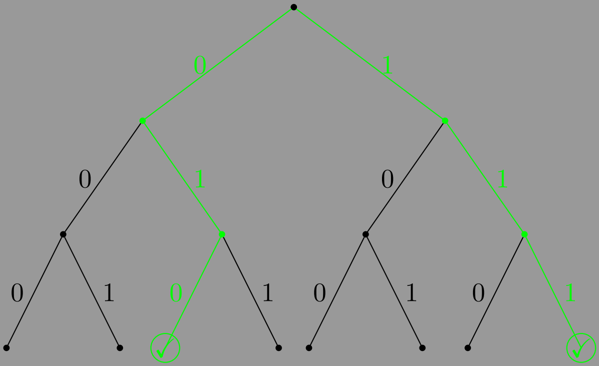
Proof: See Pr.9.1 (p.181) in [1]

($\Leftarrow$) If such an R exists, we can construct the following NTM deciding L:
"On input x, *guess* a y, such that $|y| \leq |x|^k$, and then test (in poly-time) if $(x, y) \in R$. If so, accept, else reject." Observe that an accepting computation exists if and only if $x \in L$.

($\Rightarrow$) If $L \in$ **NP**, then $\exists$ an NTM N that decides L in time $|x|^k$, for some k. Define the following R:
"$(x, y) \in R$ if and only if y is an **encoding** of an accepting computation of $N(x)$."
R is polynomially balanced and decidable (*why?*), so, given by assumption that N decides L, we have our conclusion. $\square$

Certificate Characterization of NP

Example (Encoding of a computation path)



- 010 and 111 encode accepting paths.

Can creativity be automated?

As we saw:

- Class **P**: Efficient **Computation**
- Class **NP**: Efficient **Verification**
- So, if we can efficiently verify a mathematical proof, can we create it efficiently?

Can creativity be automated?

As we saw:

- Class **P**: Efficient **Computation**
- Class **NP**: Efficient **Verification**
- So, if we can efficiently verify a mathematical proof, can we create it efficiently?

If $P = NP$...

- For every mathematical statement, and given a page limit, we would (quickly) generate a proof, if one exists.
- Given detailed constraints on an engineering task, we would (quickly) generate a design which meets the given criteria, if one exists.
- Given data on some phenomenon and modeling restrictions, we would (quickly) generate a theory to explain the data, if one exists.

See "A. Wigderson: Knowledge, Creativity and **P** versus **NP**"

Complementary complexity classes

- Deterministic complexity classes are in general closed under complement ($co\mathbf{L} = \mathbf{L}$, $co\mathbf{P} = \mathbf{P}$, $co\mathbf{PSPACE} = \mathbf{PSPACE}$).
- Complementaries of non-deterministic complexity classes are very interesting:
- The class *co***NP** contains all the languages that have **succinct disqualifications** (the analogue of *succinct certificate* for the class **NP**). The "no" instance of a problem in *co***NP** has a short proof of its being a "no" instance.
- So:

$$\boxed{\mathbf{P} \subseteq \mathbf{NP} \cap co\mathbf{NP}}$$

- Note the *similarity* and the *difference* with $\mathbf{R} = \mathbf{RE} \cap co\mathbf{RE}$.

Quantifier Characterization of Complexity Classes

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x, y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x, y)$

Quantifier Characterization of Complexity Classes

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x,y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x,y)$

- $\mathbf{P} = (\forall/\forall)$
- $\mathbf{NP} = (\exists/\forall)$
- $co\mathbf{NP} = (\forall/\exists)$

Savitch's Theorem

- REACHABILITY $\in$ **NL**.

See Ex.2.10 (p.48) in [1]

Savitch's Theorem

- REACHABILITY $\in$ **NL**.

See Ex.2.10 (p.48) in [1]

Theorem (Savitch's Theorem)

REACHABILITY $\in \mathbf{DSPACE}[\log^2 n]$

Savitch's Theorem

- REACHABILITY $\in$ **NL**. See Ex.2.10 (p.48) in [1]

Theorem (Savitch's Theorem)

REACHABILITY $\in$ **DSPACE**$[\log^2 n]$

Proof: See Th.7.4 (p.149) in [1]

$REACH(x, y, i)$: "*There is a path from x to y, of length $\leq i$*".

- We can solve REACHABILITY if we can compute $REACH(x, y, n)$, for any nodes $x, y \in V$, since any path in G can be at most n long.
- If $i = 1$, we can check whether $REACH(x, y, i)$.
- If $i > 1$, we use recursion:

Proof (*cont'd*):

```
def REACH(s,t,k)
    if k==1:
        if (s==t or (s,t) in edges): return true
    if k>1:
        for u in vertices:
            if (REACH(s,u, floor(k/2)) and
            (REACH(u,t,ceil(k/2))): return true
    return false
```

Proof (*cont'd*):

```
def REACH(s,t,k)
    if k==1:
        if (s==t or (s,t) in edges): return true
    if k>1:
        for u in vertices:
            if (REACH(s,u, floor(k/2)) and
            (REACH(u,t,ceil(k/2)))): return true
    return false
```

- We generate all nodes u one after the other, *reusing* space.
- The algorithm has recursion depth of $\lceil \log n \rceil$.
- For each recursion level, we have to store s, t, k and u, that is, $\mathcal{O}(\log n)$ space.
- Thus, the total space used is $\mathcal{O}(\log^2 n)$. $\square$

Savitch's Theorem

Corollary

$\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DSPACE}[S^2(n)]$, for any space-constructible function $S(n) \geq \log n$.

Savitch's Theorem

Corollary

$\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DSPACE}[S^2(n)]$, for any space-constructible function $S(n) \geq \log n$.

Proof:

- Let M be the nondeterministic TM to be simulated.
- We run the algorithm of Savitch's Theorem proof on the configuration graph of M on input x.
- Since the configuration graph has $c^{S(n)}$ nodes, $\mathcal{O}\left(S^2(n)\right)$ space suffices. $\square$

Savitch's Theorem

Corollary

NSPACE$[S(n)] \subseteq$ **DSPACE**$[S^2(n)]$, for any space-constructible function $S(n) \geq \log n$.

Proof:

- Let M be the nondeterministic TM to be simulated.
- We run the algorithm of Savitch's Theorem proof on the configuration graph of M on input x.
- Since the configuration graph has $c^{S(n)}$ nodes, $\mathcal{O}\left(S^2(n)\right)$ space suffices. $\square$

Corollary

$$\mathbf{PSPACE} = \mathbf{NPSPACE}$$

NL-Completeness

- In Complexity Theory, we “connect” problems in a complexity class with partial ordering relations, called **reductions**, which formalize the notion of “*a problem that is at least as hard as another*”.
- A reduction must be computationally weaker than the class in which we use it.

NL-Completeness

- In Complexity Theory, we "connect" problems in a complexity class with partial ordering relations, called **reductions**, which formalize the notion of "*a problem that is at least as hard as another*".
- A reduction must be computationally weaker than the class in which we use it.

Definition

A language L_1 is **logspace reducible** to a language L_2, denoted $L_1 \leq_m^\ell L_2$, if there is a function $f : \Sigma^* \to \Sigma^*$, computable by a DTM in $\mathcal{O}(\log n)$ space, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

We say that a language A is **NL**-complete if it is in **NL** and for every $B \in \mathbf{NL}$, $B \leq_m^\ell A$.

NL-Completeness

Theorem

REACHABILITY is **NL***-complete.*

NL-Completeness

Theorem

REACHABILITY is **NL***-complete.*

Proof: See Th.4.18 (p.89) in [2]

- We 've argued why REACHABILITY $\in$ **NL**.
- Let $L \in$ **NL**, that is, it is decided by a $\mathcal{O}(\log n)$ NTM N.
- Given input x, we can construct the *configuration graph* of $N(x)$.
- We can assume that this graph has a *single* accepting node.
- We can construct this in logspace: Given configurations C, C' we can in space $\mathcal{O}(|C| + |C'|) = \mathcal{O}(\log |x|)$ check the graph's adjacency matrix if they are connected by an edge.
- It is clear that $x \in L$ if and only if the produced instance of REACHABILITY has a "yes" answer. $\square$

Certificate Definition of NL

- We want to give a characterization of **NL**, similar to the one we gave for **NP**.
- A certificate may be polynomially long, so a logspace machine may not have the space to store it.
- So, we will assume that the certificate is provided to the machine on a separate tape that is **read once**.

Certificate Definition of NL

Definition

A language L is in **NL** if there exists a deterministic TM M with an additional special read-once input tape, such that for every $x \in \Sigma^*$:

$$x \in L \Leftrightarrow \exists y, |y| \in poly(|x|), M(x, y) = 1$$

where by $M(x, y)$ we denote the output of M where x is placed on its input tape, and y is placed on its special read-once tape, and M uses at most $\mathcal{O}(\log |x|)$ space on its read-write tapes for every input x.

- What if remove the read-once restriction and allow the TM's head to move back and forth on the certificate, and read each bit multiple times?

Immerman-Szelepscényi

Theorem (The Immerman-Szelepscényi Theorem)

$\overline{\text{REACHABILITY}} \in \mathbf{NL}$

Immerman-Szelepscényi

Theorem (The Immerman-Szelepscényi Theorem)

$\overline{\text{REACHABILITY}} \in \mathbf{NL}$

Proof: See Th.4.20 (p.91) in [2]

- It suffices to show a $\mathcal{O}(\log n)$ verification algorithm A such that: $\forall\ (G, s, t)$, $\exists$ a polynomial certificate u such that: $A((G, s, t), u) =$ "yes" iff t is not reachable from s.
- A has read-once access to u.
- G's vertices are identified by numbers in $\{1, \ldots, n\} = [n]$
- C_i: "*The set of vertices reachable from s in* $\leq i$ *steps.*"
- Membership in C_i is easily certified:
- $\forall i \in [n]$: $v_0, \ldots, v_k$ along the path from s to v, $k \leq i$.
- The certificate is at most polynomial in n.

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

- We can check the certificate using read-once access:
  1. $v_0 = s$
  2. for $j > 0$, $(v_{j-1}, v_j) \in E(G)$
  3. $v_k = v$
  4. Path ends within at most i steps
- We now construct two types of certificates:
  1. A certificate that a vertex $v \notin C_i$, given $|C_i|$.
  2. A certificate that $|C_i| = c$, for some c, given $|C_{i-1}|$.
- Since $C_0 = \{s\}$, we can provide the 2nd certificate to convince the verifier for the sizes of $C_1, \dots, C_n$
- C_n is the set of vertices *reachable* from s.

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

- Since the verifier has been convinced of $|C_n|$, we can use the 1st type of certificate to convince the verifier that $t \notin C_n$.

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

- Since the verifier has been convinced of $|C_n|$, we can use the 1st type of certificate to convince the verifier that $t \notin C_n$.

- **Certifying that $v \notin C_i$, given $|C_i|$**
  The certificate is the list of certificates that $u \in C_i$, for every $u \in C_i$.
  The verifier will check:
  1. Each certificate is valid
  2. Vertex u, given a certificate for u, is larger than the previous.
  3. No certificate is provided for v.
  4. The total number of certificates is exactly $|C_i|$.

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

Certifying that $v \notin C_i$, given $|C_{i-1}|$

The certificate is the list of certificates that $u \in C_{i-1}$, for every $u \in C_{i-1}$

The verifier will check:

1. Each certificate is valid
2. Vertex u, given a certificate for u, is larger than the previous.
3. No certificate is provided for v *or for a neighbour of* v.
4. The total number of certificates is exactly $|C_{i-1}|$.

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

Certifying that $v \notin C_i$, given $|C_{i-1}|$

The certificate is the list of certificates that $u \in C_{i-1}$, for every $u \in C_{i-1}$

The verifier will check:

1. Each certificate is valid
2. Vertex u, given a certificate for u, is larger than the previous.
3. No certificate is provided for v *or for a neighbour of* v.
4. The total number of certificates is exactly $|C_{i-1}|$.

Certifying that $|C_i| = c$, given $|C_{i-1}|$

The certificate will consist of n certificates, for vertices 1 to n, in ascending order.

The verifier will check all certificates, and count the vertices that have been certified to be in C_i. If $|C_i| = c$, it accepts. $\square$

The Immerman-Szelepscényi Theorem

Corollary

For every space constructible $S(n) > \log n$:

$$\mathbf{NSPACE}[S(n)] = co\mathbf{NSPACE}[S(n)]$$

Proof:

- Let $L \in \mathbf{NSPACE}[S(n)]$. We will show that $\exists\ S(n)$ space-bounded NTM $\overline{M}$ deciding $\overline{L}$:
- $\overline{M}$ on input x uses the above certification procedure on the *configuration graph* of M. $\square$

The Immerman-Szelepscényi Theorem

Corollary

For every space constructible $S(n) > \log n$:

$$\mathbf{NSPACE}[S(n)] = co\mathbf{NSPACE}[S(n)]$$

Proof:

- Let $L \in \mathbf{NSPACE}[S(n)]$. We will show that $\exists$ $S(n)$ space-bounded NTM $\overline{M}$ deciding $\overline{L}$:
- $\overline{M}$ on input x uses the above certification procedure on the *configuration graph* of M. $\square$

Corollary

$$\mathbf{NL} = co\mathbf{NL}$$

What about Undirected Reachability?

- UNDIRECTED REACHABILITY captures the phenomenon of configuration graphs with both directions.
- H. Lewis and C. Papadimitriou defined the class **SL** (**S**ymmetric **L**ogspace) as the class of languages decided by a **Symmetric Turing Machine** using logarithmic space.
- Obviously,

$$\boxed{\mathbf{L} \subseteq \mathbf{SL} \subseteq \mathbf{NL}}$$

- As in the case of **NL**, UNDIRECTED REACHABILITY is **SL**-complete.
- But in 2004, Omer Reingold showed, using expander graphs, a deterministic logspace algorithm for UNDIRECTED REACHABILITY, so:

What about Undirected Reachability?

- UNDIRECTED REACHABILITY captures the phenomenon of configuration graphs with both directions.
- H. Lewis and C. Papadimitriou defined the class **SL** (**S**ymmetric **L**ogspace) as the class of languages decided by a **Symmetric Turing Machine** using logarithmic space.
- Obviously,

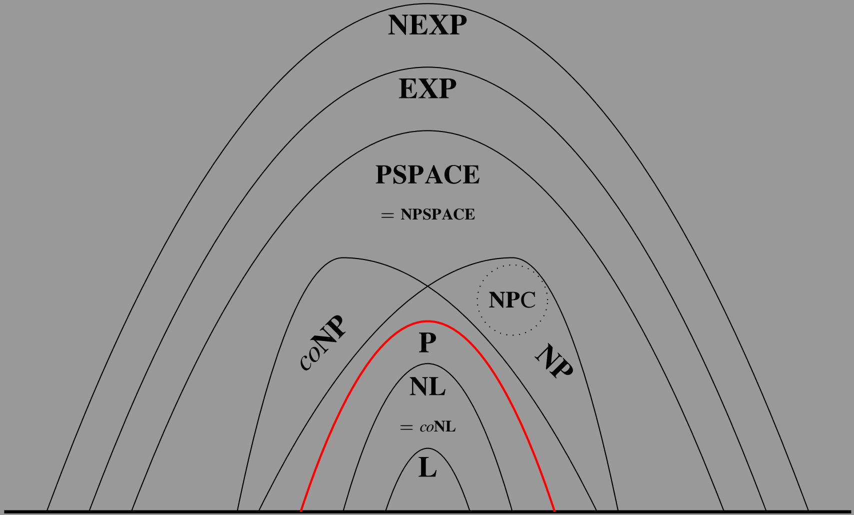
$$\boxed{\mathbf{L} \subseteq \mathbf{SL} \subseteq \mathbf{NL}}$$

- As in the case of **NL**, UNDIRECTED REACHABILITY is **SL**-complete.
- But in 2004, Omer Reingold showed, using expander graphs, a deterministic logspace algorithm for UNDIRECTED REACHABILITY, so:

Theorem (Reingold, 2004)

$$\mathbf{L} = \mathbf{SL}$$

Our Complexity Hierarchy Landscape



Karp Reductions

Definition

A language L_1 is **Karp reducible** to a language L_2, denoted by $L_1 \leq_m^p L_2$, if there is a function $f : \Sigma^* \to \Sigma^*$, computable by a polynomial-time DTM, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Karp Reductions

Definition

A language L_1 is **Karp reducible** to a language L_2, denoted by $L_1 \leq_m^p L_2$, if there is a function $f : \Sigma^* \to \Sigma^*$, computable by a polynomial-time DTM, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Definition

Let $\mathcal{C}$ be a complexity class.

- We say that a language A is **$\mathcal{C}$-hard** (or $\leq_m^p$-hard for $\mathcal{C}$) if for every $B \in \mathcal{C}$, $B \leq_m^p A$.
- We say that a language A is **$\mathcal{C}$-complete**, if it is $\mathcal{C}$-hard, and also $A \in \mathcal{C}$.

Karp reductions vs logspace redutions

Theorem

A logspace reduction is a polynomial-time reduction.

Proof: See Th.8.1 (p.160) in [1]

- Let M the logspace reduction TM.
- M has $2^{\mathcal{O}(\log n)}$ possible configurations.
- The machine is deterministic, so *no configuration can be repeated* in the computation.
- So, the computation takes $\mathcal{O}\left(n^k\right)$ time, for some k. $\square$

Circuits and CVP

Definition (Boolean circuits)

For every $n \in \mathbb{N}$ an n-input, single output Boolean Circuit C is a directed acyclic graph with n sources and *one* sink.

- All nonsource vertices are called *gates* and are labeled with one of $\wedge$ (and), $\vee$ (or) or $\neg$ (not).
- The vertices labeled with $\wedge$ and $\vee$ have *fan-in* (i.e. number or incoming edges) 2.
- The vertices labeled with $\neg$ have *fan-in* 1.
- For every vertex v of C, we assign a value as follows: for some input $x \in \{0,1\}^n$, if v is the i-th input vertex then $val(v) = x_i$, and otherwise $val(v)$ is defined recursively by applying v's logical operation on the values of the vertices connected to v.
- The *output* $C(x)$ is the value of the output vertex.

Circuits and CVP

Definition (CVP)

Circuit Value Problem (CVP): Given a circuit C and an assignment x to its variables, determine whether $C(x) = 1$.

- CVP $\in \mathbf{P}$.

Circuits and CVP

Definition (CVP)

Circuit Value Problem (CVP): Given a circuit C and an assignment x to its variables, determine whether $C(x) = 1$.

- CVP $\in$ **P**.

Example

REACHABILITY $\leq_m^\ell$ CVP: Graph $G \rightarrow$ circuit $R(G)$:

- The gates are of the form:
  - $g_{i,j,k}$, $1 \leq i, j \leq n$, $0 \leq k \leq n$.
  - $h_{i,j,k}$, $1 \leq i, j, k \leq n$
- $g_{i,j,k}$ is **true** iff there is a path from i to j without intermediate nodes bigger than k.
- $h_{i,j,k}$ is **true** iff there is a path from i to j without intermediate nodes bigger than k, and k is used.

Circuits and CVP

Example

- Input gates: $g_{i,j,0}$ is **true** iff ($i = j$ or $(i,j) \in E(G)$).
- For $k = 1, \ldots, n$: $h_{i,j,k} = (g_{i,k,k-1} \wedge g_{k,j,k-1})$
- For $k = 1, \ldots, n$: $g_{i,j,k} = (g_{i,j,k-1} \vee h_{i,j,k})$
- The output gate $g_{1,n,n}$ is **true** iff there is a path from 1 to n using no intermediate paths above n (sic).
- We also can compute the reduction in logspace: go over all possible i, j, k's and output the appropriate edges and sorts for the variables $(1, \ldots, 2n^3 + n^2)$.

Composing Reductions

Theorem

If $L_1 \leq_m^\ell L_2$ and $L_2 \leq_m^\ell L_3$, then $L_1 \leq_m^\ell L_3$.

Proof:

See Prop.8.2 (p.164) in [1]

- Let R, R' be the aforementioned reductions.
- We have to prove that $R'(R(x))$ is a logspace reduction.
- But $R(x)$ may by longer than $\log |x|$...

Composing Reductions

Theorem

If $L_1 \leq_m^\ell L_2$ and $L_2 \leq_m^\ell L_3$, then $L_1 \leq_m^\ell L_3$.

Proof: See Prop.8.2 (p.164) in [1]

- Let R, R' be the aforementioned reductions.
- We have to prove that $R'(R(x))$ is a logspace reduction.
- But $R(x)$ may by longer than $\log |x|$...
- We simulate $M_{R'}$ by remembering the head position i of the input string of $M_{R'}$, i.e. the output string of M_R.
- If the head moves to the right, we increment i and simulate M_R long enough to take the i^{th} bit of the output.
- If the head stays in the same position, we just remember the i^{th} bit.
- If the head moves to the left, we decrement i and **start M_R from the beginning**, until we reach the desired bit. $\square$

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class $\mathcal{C}$ is **closed under reductions** if for all $A, B \subseteq \Sigma^*$:
If $A \leq B$ and $B \in \mathcal{C}$, then $A \in \mathcal{C}$.

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class $\mathcal{C}$ is **closed under reductions** if for all $A, B \subseteq \Sigma^*$:
If $A \leq B$ and $B \in \mathcal{C}$, then $A \in \mathcal{C}$.

- **P**, **NP**, *co***NP**, **L**, **NL**, **PSPACE**, **EXP** are closed under Karp and logspace reductions.

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class $\mathcal{C}$ is **closed under reductions** if for all $A, B \subseteq \Sigma^*$:
If $A \leq B$ and $B \in \mathcal{C}$, then $A \in \mathcal{C}$.

- **P**, **NP**, *co***NP**, **L**, **NL**, **PSPACE**, **EXP** are closed under Karp and logspace reductions.
- If an **NP**-complete language is in **P**, then **P** $=$ **NP**.

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class $\mathcal{C}$ is **closed under reductions** if for all $A, B \subseteq \Sigma^*$:
If $A \leq B$ and $B \in \mathcal{C}$, then $A \in \mathcal{C}$.

- **P**, **NP**, *co***NP**, **L**, **NL**, **PSPACE**, **EXP** are closed under Karp and logspace reductions.
- If an **NP**-complete language is in **P**, then $\mathbf{P} = \mathbf{NP}$.
- If L is **NP**-complete, then $\bar{L}$ is *co***NP**-complete.

Closure under reductions

- Complete problems are the **maximal elements** of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class $\mathcal{C}$ is **closed under reductions** if for all $A, B \subseteq \Sigma^*$:
If $A \leq B$ and $B \in \mathcal{C}$, then $A \in \mathcal{C}$.

- $\mathbf{P}, \mathbf{NP}, co\mathbf{NP}, \mathbf{L}, \mathbf{NL}, \mathbf{PSPACE}, \mathbf{EXP}$ are closed under Karp and logspace reductions.
- If an **NP**-complete language is in **P**, then $\mathbf{P} = \mathbf{NP}$.
- If L is **NP**-complete, then $\bar{L}$ is *co***NP**-complete.
- If a *co***NP**-complete problem is in **NP**, then $\mathbf{NP} = co\mathbf{NP}$.

P-Completeness

Theorem

If two classes $\mathcal{C}$ and $\mathcal{C}'$ are both closed under reductions and there is an $L \subseteq \Sigma^*$ complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.

P-Completeness

Theorem

If two classes $\mathcal{C}$ and $\mathcal{C}'$ are both closed under reductions and there is an $L \subseteq \Sigma^*$ complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.

- Consider the **Computation Table** T of a poly-time TM $M(x)$:
  $\boxed{T_{ij} \text{ represents the contents of tape position } j \text{ at step } i.}$
- But how to remember the head position and state?
  At the i^{th} step: if the state is q and the head is in position j, then $T_{ij} \in \Sigma \times Q$.
- We say that the table is **accepting** if $T_{|x|^k-1,j} \in (\Sigma \times \{q_{yes}\})$, for some j.
- Observe that T_{ij} depends only on the contents of the **same** of **adjacent** positions at time $i - 1$.

P-Completeness

Theorem

CVP is **P***-complete.*

P-Completeness

Theorem

CVP is **P***-complete.*

Proof: See Th. 8.1 (p.168) in [1]

- We have to show that for any $L \in \mathbf{P}$ there is a reduction R from L to CVP.
- $R(x)$ must be a variable-free circuit such that $x \in L \Leftrightarrow R(x) = 1$.
- T_{ij} depends only on $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$.
- Let $\Gamma = \Sigma \cup (\Sigma \times Q)$.
- Encode $s \in \Gamma$ as $(s_1, \ldots, s_m)$, where $m = \lceil \log |\Gamma| \rceil$.
- Then the computation table can be seen as a table of binary entries $S_{ij\ell}$, $1 \leq \ell \leq m$.
- $S_{ij\ell}$ depends only on the $3m$ entries $S_{i-1,j-1,\ell'}, S_{i-1,j,\ell'}, S_{i-1,j+1,\ell'}$, where $1 \leq \ell' \leq m$.

P-Completeness

Proof (*cont'd*):

- So, there are m Boolean Functions $f_1, \ldots, f_m : \{0,1\}^{3m} \to \{0,1\}$ s.t.:
$$S_{ij\ell} = f_\ell(\overrightarrow{S}_{i-1,j-1}, \overrightarrow{S}_{i-1,j}, \overrightarrow{S}_{i-1,j+1})$$
- Thus, there exists a Boolean Circuit C with $3m$ inputs and m outputs computing T_{ij}.
- C depends only on M, and has constant size.
- $R(x)$ will be $(|x|^k - 1) \times (|x|^k - 2)$ copies of C.
- The input gates are fixed.
- $R(x)$'s output gate will be the first bit of $C_{|x|^k-1,1}$.
- The circuit C is fixed, so we can generate indexed copies of C, using $\mathcal{O}(\log|x|)$ space for indexing. $\square$

CIRCUIT SAT & SAT

Definition (CIRCUIT SAT)

Given Boolen Circuit C, is there a truth assignment x appropriate to C, such that $C(x) = 1$?

Definition (SAT)

Given a Boolean Expression ϕ in CNF, is it satisfiable?

CIRCUIT SAT & SAT

Definition (CIRCUIT SAT)

Given Boolen Circuit C, is there a truth assignment x appropriate to C, such that $C(x) = 1$?

Definition (SAT)

Given a Boolean Expression ϕ in CNF, is it satisfiable?

Example

CIRCUIT SAT $\leq_m^\ell$ SAT:

- Given $C \rightarrow$ Boolean Formula $R(C)$, s.t. $C(x) = 1 \Leftrightarrow R(C)(x) = T$.
- Variables of $C \rightarrow$ variables of $R(C)$.
- Gate g of $C \rightarrow$ variable g of $R(C)$.

CIRCUIT SAT & SAT

Example

- Gate g of $C \to$ clauses in $R(C)$:
  - g **variable** gate: add $(\neg g \vee x) \wedge (g \vee \neg x)$ $\equiv g \Leftrightarrow x$
  - g **TRUE** gate: add (g)
  - g **FALSE** gate: add $(\neg g)$
  - g **NOT** gate & $pred(g) = h$: add $(\neg g \vee \neg h) \wedge (g \vee h)$ $\equiv g \Leftrightarrow \neg h$
  - g **OR** gate & $pred(g) = \{h, h'\}$: add
    $(\neg h \vee g) \wedge (\neg h' \vee g) \wedge (h \vee h' \vee \neg g)$ $\equiv g \Leftrightarrow (h \vee h')$
  - g **AND** gate & $pred(g) = \{h, h'\}$: add
    $(\neg g \vee h) \wedge (\neg g \vee h') \wedge (\neg h \vee \neg h' \vee g)$ $\equiv g \Leftrightarrow (h \wedge h')$
  - g **output** gate: add (g)
- $R(C)$ is satisfiable if and only if C is.
- The construction can be done within $\log |x|$ space. □

Bounded Halting Problem

- We can define the time-bounded analogue of HP:

Definition (Bounded Halting Problem (BHP))

Given the code $\llcorner M \lrcorner$ of an NTM M, and input x and a string 0^t, decide if M accepts x in t steps.

Bounded Halting Problem

- We can define the time-bounded analogue of HP:

Definition (Bounded Halting Problem (BHP))

Given the code $\llcorner M \lrcorner$ of an NTM M, and input x and a string 0^t, decide if M accepts x in t steps.

Theorem

BHP is **NP***-complete.*

Bounded Halting Problem

- We can define the time-bounded analogue of HP:

Definition (Bounded Halting Problem (BHP))

Given the code $\llcorner M \lrcorner$ of an NTM M, and input x and a string 0^t, decide if M accepts x in t steps.

Theorem

BHP is **NP***-complete.*

Proof:

- BHP $\in$ **NP**.
- Let $A \in$ **NP**. Then, $\exists$ NTM M deciding A in time $p(|x|)$, for some $p \in poly(|x|)$.
- The reduction is the function $R(x) = \langle \llcorner M \lrcorner, x, 0^{p(|x|)} \rangle$. $\square$

Cook’s Theorem

Theorem (Cook’s Theorem)

SAT is **NP***-complete.*

Cook's Theorem

Theorem (Cook's Theorem)

SAT is **NP***-complete.*

Proof: See Th.8.2 (p.171) in [1]

- SAT $\in$ **NP**.
- Let $L \in$ **NP**. We will show that $L \leq_m^\ell$ CIRCUIT SAT $\leq_m^\ell$ SAT.
- Since $L \in$ **NP**, there exists an NPTM M deciding L in n^k steps.
- Let $(c_1, \dots, c_{n^k}) \in \{0,1\}^{n^k}$ a certificate for M (recall the binary encoding of the computation tree).

Cook's Theorem

Proof (*cont'd*):

- If we fix a certificate, then the computation is *deterministic* (the language's Verifier $V(x, y)$ is a DPTM).
- So, we can define the **computation table** $T(M, x, \overrightarrow{c})$.
- As before, all non-top row and non-extreme column cells T_{ij} will depend *only* on $T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}$ and the nondeterministic choice c_{i-1}.
- We now fixed a circuit C with $3m + 1$ input gates.
- Thus, we can construct in $\log |x|$ space a circuit $R(x)$ with variable gates $c_1, \ldots c_{n^k}$ corresponding to the **nondeterministic choices** of the machine.
- $R(x)$ is satisfiable if and only if $x \in L$. $\square$

NP-completeness: Web of Reductions

- Many **NP**-complete problems stem from Cook's Theorem via reductions:
  - 3SAT, MAX2SAT, NAESAT
  - IS, CLIQUE, VERTEX COVER, MAX CUT
  - $\text{TSP}_{(\text{D})}$, 3COL
  - SET COVER, PARTITION, KNAPSACK, BIN PACKING
  - INTEGER PROGRAMMING (IP): Given m inequalities oven n variables $u_i \in \{0, 1\}$, is there an assignment satisfying all the inequalities?
- Always remember that these are **decision versions** of the corresponding **optimization problems**.

NP-completeness: Web of Reductions

- Many **NP**-complete problems stem from Cook's Theorem via reductions:
  - 3SAT, MAX2SAT, NAESAT
  - IS, CLIQUE, VERTEX COVER, MAX CUT
  - $\text{TSP}_{(\text{D})}$, 3COL
  - SET COVER, PARTITION, KNAPSACK, BIN PACKING
  - INTEGER PROGRAMMING (IP): Given m inequalities oven n variables $u_i \in \{0, 1\}$, is there an assignment satisfying all the inequalities?
- Always remember that these are **decision versions** of the corresponding **optimization problems**.
- But 2SAT, 2COL $\in$ **P**.

NP-completeness: Web of Reductions

Example

SAT $\leq_m^\ell$ IP:

- Every clause can be expressed as an inequality, eg:

$$(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \longrightarrow x_1 + (1 - x_2) + (1 - x_3) \geq 1$$

NP-completeness: Web of Reductions

Example

SAT $\leq_m^{\ell}$ IP:

- Every clause can be expressed as an inequality, eg:

$$(x_1 \lor \bar{x}_2 \lor \bar{x}_3) \longrightarrow x_1 + (1 - x_2) + (1 - x_3) \geq 1$$

- This method is generalized by the notion of *Constraint Satisfaction Problems*.
- A **Constraint Satisfaction Problem** (CSP) generalizes SAT by allowing clauses of arbitrary form (instead of ORs of literals).

3SAT is the subcase of *qCSP*, where arity $q = 3$ and the constraints are ORs of the involved literals.

Quantified Boolean Formulas

Definition (Quantified Boolean Formula)

A **Quantified Boolean Formula** F is a formula of the form:

$$F = \exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \; \phi(x_1, \ldots, x_n)$$

where ϕ is *plain* (quantifier-free) boolean formula.

- Let TQBF the language of all true QBFs.

Quantified Boolean Formulas

Definition (Quantified Boolean Formula)

A **Quantified Boolean Formula** F is a formula of the form:

$$F = \exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \; \phi(x_1, \ldots, x_n)$$

where ϕ is *plain* (quantifier-free) boolean formula.

- Let TQBF the language of all true QBFs.

Example

$$F = \exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \right]$$

The above is a True QBF ($(1,0,0)$ and $(1,1,1)$ satisfy it).

Quantified Boolean Formulas

Theorem

TQBF *is* **PSPACE***-complete.*

Quantified Boolean Formulas

Theorem

TQBF *is* **PSPACE**-*complete.*

Proof: See Th. 19.1 (p.456) in [1] – Th.4.13 (p.84) in [2]

- TQBF $\in$ **PSPACE**:
  - Let ϕ be a QBF, with n variables and length m.
  - Recursive algorithm $A(\phi)$:
  - If $n = 0$, then there are only constants, hence $\mathcal{O}(m)$ time/space.
  - If $n > 0$:
    $A(\phi) = A\left(\phi|_{x_1=0}\right) \vee A\left(\phi|_{x_1=1}\right)$, if $Q_1 = \exists$, and
    $A(\phi) = A\left(\phi|_{x_1=0}\right) \wedge A\left(\phi|_{x_1=1}\right)$, if $Q_1 = \forall$.
  - Both recursive computations can be run on *the same space*.
  - So $space_{n,m} = space_{n-1,m} + \mathcal{O}(m) \Rightarrow space_{n,m} = \mathcal{O}(n \cdot m)$.

Quantified Boolean Formulas

Proof (*cont'd*):

- Now, let M a TM with space bound $p(n)$.
- We can create the configuration graph of $M(x)$, having size $2^{\mathcal{O}(p(n))}$.
- M accepts x iff there is a path of length at most $2^{\mathcal{O}(p(n))}$ from the initial to the accepting configuration.
- Using Savitch's Theorem idea, for two configurations C and C' we have:
  $REACH(C, C', 2^i) \Leftrightarrow$
  $\Leftrightarrow \exists C'' \left[REACH(C, C'', 2^{i-1}) \wedge REACH(C'', C', 2^{i-1}) \right]$

Quantified Boolean Formulas

Proof (*cont'd*):

- Now, let M a TM with space bound $p(n)$.
- We can create the configuration graph of $M(x)$, having size $2^{\mathcal{O}(p(n))}$.
- M accepts x iff there is a path of length at most $2^{\mathcal{O}(p(n))}$ from the initial to the accepting configuration.
- Using Savitch's Theorem idea, for two configurations C and C' we have:
  $REACH(C, C', 2^i) \Leftrightarrow$
  $\Leftrightarrow \exists C'' \left[REACH(C, C'', 2^{i-1}) \wedge REACH(C'', C', 2^{i-1}) \right]$
- But, this is a bad idea: Doubles the size each time.
- Instead, we use additional variables:
  $\exists C'' \forall D_1 \forall D_2 \left[(D_1 = C \wedge D_2 = C'') \vee (D_1 = C'' \wedge D_2 = C') \right] \Rightarrow REACH(D_1, D_2, 2^{i-1})$

Quantified Boolean Formulas

Proof (*cont'd*):

- The base case of the recursion is $C_1 \to C_2$, and can be encoded as a quantifier-free formula.
- The size of the formula in the i^{th} step is $space_i \leq space_{i-1} + \mathcal{O}(p(n)) \Rightarrow \mathcal{O}\left(p^2(n)\right)$. $\square$

*Logical Characterizations

- **Descriptive complexity** is a branch of computational complexity theory and of finite model theory that characterizes complexity classes by the *type of logic* needed to express the languages in them.

*Logical Characterizations

- **Descriptive complexity** is a branch of computational complexity theory and of finite model theory that characterizes complexity classes by the *type of logic* needed to express the languages in them.

Theorem (Fagin's Theorem)

The set of all properties expressible in Existential Second-Order Logic is precisely **NP**.

Theorem

The class of all properties expressible in Horn Existential Second-Order Logic with Successor is precisely **P**.

- HORNSAT is **P**-complete.

Summary 1/2

- We define complexity classes using a computation model/mode and complexity measures.
- Time/Space constructible functions are used as complexity measures.
- Classes of the same kind form *proper hierarchies*.
- **NP** is the class of *easily verifiable* problems: given a *certificate*, one can efficiently verify that it is correct.
- Savitch's Theorem implies that **PSPACE** = **NPSPACE**.

Summary 2/2

- Reductions relate problems with respect to hardness.
- Complete problems reflect the difficulty of the class.
- REACHABILITY is **NL**-complete.
- Immerman-Szelepscényi's Theorem implies that $\mathbf{NL} = co\mathbf{NL}$.
- Circuit Value Problem (CVP) is **P**-complete under logspace reductions.
- CIRCUIT SAT and SAT are **NP**-complete.
- True Quantified Boolean Formula (TQBF) is **PSPACE**-complete.

Contents

Oracle TMs and Oracle Classes

Definition

A Turing Machine $M^?$ with *oracle* is a multi-string deterministic TM that has a special string, called **query string**, and three special states: $q_?$ (**query state**), and q_{YES}, q_{NO} (*answer states*). Let $A \subseteq \Sigma^*$ be an arbitrary language. The computation of oracle machine M^A proceeds like an ordinary TM except for transitions from the query state: *From the $q_?$ moves to either q_{YES}, q_{NO}, depending on whether the current query string is in A or not.*

Oracle TMs and Oracle Classes

Definition

A Turing Machine $M^?$ with *oracle* is a multi-string deterministic TM that has a special string, called **query string**, and three special states: $q_?$ (**query state**), and q_{YES}, q_{NO} (*answer states*). Let $A \subseteq \Sigma^*$ be an arbitrary language. The computation of oracle machine M^A proceeds like an ordinary TM except for transitions from the query state: *From the $q_?$ moves to either q_{YES}, q_{NO}, depending on whether the current query string is in A or not.*

- The answer states allow the machine to use this answer to its further computation.
- The computation of $M^?$ with oracle A on iput x is denoted as $M^A(x)$.

Oracle TMs and Oracle Classes

Definition

Let $\mathcal{C}$ be a time complexity class (deterministic or nondeterministic). Define $\mathcal{C}^A$ to be the *class* of all languages decided by machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now oracle access to A. Also, we define: $\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{L \in \mathcal{C}_2} \mathcal{C}_1^L$.

For example, $\mathbf{P}^{\mathbf{NP}} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^L$. Note that $\mathbf{P}^{\mathrm{SAT}} = \mathbf{P}^{\mathbf{NP}}$.

Oracle TMs and Oracle Classes

Definition

Let $\mathcal{C}$ be a time complexity class (deterministic or nondeterministic). Define $\mathcal{C}^A$ to be the *class* of all languages decided by machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now oracle access to A. Also, we define: $\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{L \in \mathcal{C}_2} \mathcal{C}_1^L$.

For example, $\mathbf{P}^{\mathbf{NP}} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^L$. Note that $\mathbf{P}^{\text{SAT}} = \mathbf{P}^{\mathbf{NP}}$.

Theorem

There exists an oracle A for which $\mathbf{P}^A = \mathbf{NP}^A$.

Oracle TMs and Oracle Classes

Definition

Let $\mathcal{C}$ be a time complexity class (deterministic or nondeterministic). Define $\mathcal{C}^A$ to be the *class* of all languages decided by machines of the same sort and time bound as in $\mathcal{C}$, only that the machines have now oracle access to A. Also, we define: $\mathcal{C}_1^{\mathcal{C}_2} = \bigcup_{L \in \mathcal{C}_2} \mathcal{C}_1^L$.

For example, $\mathbf{P}^{\mathbf{NP}} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^L$. Note that $\mathbf{P}^{\text{SAT}} = \mathbf{P}^{\mathbf{NP}}$.

Theorem

There exists an oracle A for which $\mathbf{P}^A = \mathbf{NP}^A$.

Proof: Th.14.4 (p.340) in [1]

Take A to be a **PSPACE**-complete language.Then:

$$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A = \mathbf{PSPACE}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE}. \quad \square$$

Oracle TMs and Oracle Classes

Theorem

There exists an oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$.

Oracle TMs and Oracle Classes

Theorem

There exists an oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$.

Proof: Th.14.5 (p.340-342) in [1]

- We will find a language $L \in \mathbf{NP}^B \setminus \mathbf{P}^B$.
- Let $L = \{1^n \mid \exists x \in B \text{ with } |x| = n\}$.
- $L \in \mathbf{NP}^B$ *(why?)*
- We will define the oracle $B \subseteq \{0,1\}^*$ such that $L \notin \mathbf{P}^B$:

Oracle TMs and Oracle Classes

Theorem

There exists an oracle B for which $\mathbf{P}^B \neq \mathbf{NP}^B$.

Proof: Th.14.5 (p.340-342) in [1]

- We will find a language $L \in \mathbf{NP}^B \setminus \mathbf{P}^B$.
- Let $L = \{1^n \mid \exists x \in B \text{ with } |x| = n\}$.
- $L \in \mathbf{NP}^B$ *(why?)*
- We will define the oracle $B \subseteq \{0,1\}^*$ such that $L \notin \mathbf{P}^B$:
- Let $M_1^?, M_2^?, \dots$ an enumeration of all PDTMs with oracle, such that every machine appears *infinitely many* times in the enumeration.
- We will define B iteratively: $B_0 = \emptyset$, and $B = \bigcup_{i \geq 0} B_i$.
- In i^{th} stage, we have defined B_{i-1}, the set of all strings in B with length $< i$.
- Let also X the set of **exceptions**.

Proof (*cont'd*):

- We simulate $M_i^B(1^i)$ for $i^{\log i}$ steps.
- How do we answer the oracle questions "*Is* $x \in B$"?

Proof (*cont'd*):

- We simulate $M_i^B(1^i)$ for $i^{\log i}$ steps.
- How do we answer the oracle questions "*Is* $x \in B$"?
- **If** $|x| < i$, we look for x in B_{i-1}.
- $\rightarrow$ **If** $x \in B_{i-1}$, M_i^B goes to q_{YES}
  $\rightarrow$ **Else** M_i^B goes to q_{NO}
- **If** $|x| \geq i$, M_i^B goes to q_{NO} ,and $x \rightarrow X$.

Proof (*cont'd*):

- We simulate $M_i^B(1^i)$ for $i^{\log i}$ steps.
- How do we answer the oracle questions "*Is* $x \in B$"?
- **If** $|x| < i$, we look for x in B_{i-1}.
- $\rightarrow$ **If** $x \in B_{i-1}$, M_i^B goes to q_{YES}
  $\rightarrow$ **Else** M_i^B goes to q_{NO}
- **If** $|x| \geq i$, M_i^B goes to q_{NO} ,and $x \rightarrow X$.
- Suppose that after at most $i^{\log i}$ steps the machine *rejects*.
  - Then we define $B_i = B_{i-1} \cup \{x \in \{0,1\}^* : |x| = i, x \notin X\}$ so $1^i \in L$, and $L(M_i^B) \neq L$.
  - Why $\{x \in \{0,1\}^* : |x| = i, x \notin X\} \neq \emptyset$? ?
- If the machine *accepts*, we define $B_i = B_{i-1}$, so that $1^i \notin L$.
- If the machine fails to halt in the allotted time, we set $B_i = B_{i-1}$, but we know that the same machine will appear in the enumeration with an index sufficiently large. □

A First Barrier: The Limits of Diagonalization

- As we saw, an oracle can transfer us to an alternative computational "*universe*".
  (We saw a universe where $\mathbf{P} = \mathbf{NP}$, and another where $\mathbf{P} \neq \mathbf{NP}$)
- Diagonalization is a technique that relies in the facts that:
  - **TMs are (effectively) represented by strings.**
  - **A TM can simulate another without much overhead in time/space.**

A First Barrier: The Limits of Diagonalization

- As we saw, an oracle can transfer us to an alternative computational "*universe*".
  (We saw a universe where $\mathbf{P} = \mathbf{NP}$, and another where $\mathbf{P} \neq \mathbf{NP}$)
- Diagonalization is a technique that relies in the facts that:
  - **TMs are (effectively) represented by strings.**
  - **A TM can simulate another without much overhead in time/space.**
- So, diagonalization or any other proof technique relies only on these two facts, holds also for *every* oracle.
- Such results are called **relativizing results**.
  E.g., $\mathbf{P}^A \subseteq \mathbf{NP}^A$, for every $A \in \{0,1\}^*$.
- The above two theorems indicate that **P** vs. **NP** is a **nonrelativizing** result, so diagonalization and any other relativizing method doesn't suffice to prove it.

Cook Reductions

- A problem A is **Cook-Reducible** to a problem B, denoted by $A \leq_T^p B$, if there is an oracle DTM M^B which in polynomial time decides A (*making at most polynomial many queries to* B).
- That is: $A \in \mathbf{P}^B$.

Cook Reductions

- A problem A is **Cook-Reducible** to a problem B, denoted by $A \leq_T^p B$, if there is an oracle DTM M^B which in polynomial time decides A (*making at most polynomial many queries to* B).
- That is: $A \in \mathbf{P}^B$.

- $A \leq_m^p B \Rightarrow A \leq_T^p B$
- $\overline{A} \leq_T^p A$

Cook Reductions

- A problem A is **Cook-Reducible** to a problem B, denoted by $A \leq_T^p B$, if there is an oracle DTM M^B which in polynomial time decides A (*making at most polynomial many queries to* B).
- That is: $A \in \mathbf{P}^B$.

- $A \leq_m^p B \Rightarrow A \leq_T^p B$
- $\overline{A} \leq_T^p A$

Theorem

$\mathbf{P}, \mathbf{PSPACE}$ *are closed under* $\leq_T^p$.

- Is **NP** closed under $\leq_T^p$? (*cf. Problem Sets!*)

*Random Oracles

- We proved that:
  - $\exists A \subseteq \Sigma^* : \mathbf{P}^A = \mathbf{NP}^A$
  - $\exists B \subseteq \Sigma^* : \mathbf{P}^B \neq \mathbf{NP}^B$
- What if we chose the oracle language at random?

*Random Oracles

- We proved that:
  - $\exists A \subseteq \Sigma^* : \mathbf{P}^A = \mathbf{NP}^A$
  - $\exists B \subseteq \Sigma^* : \mathbf{P}^B \neq \mathbf{NP}^B$
- What if we chose the oracle language at random?
- Now, consider the set $\mathcal{U} = Pow(\Sigma^*)$, and the sets:

$$\{A \in \mathcal{U} : \mathbf{P}^A = \mathbf{NP}^A\}$$

$$\{B \in \mathcal{U} : \mathbf{P}^B \neq \mathbf{NP}^B\}$$

- Can we compare these two sets, and find which is *larger*?

*Random Oracles

- We proved that:
  - $\exists A \subseteq \Sigma^* : \mathbf{P}^A = \mathbf{NP}^A$
  - $\exists B \subseteq \Sigma^* : \mathbf{P}^B \neq \mathbf{NP}^B$
- What if we chose the oracle language at random?
- Now, consider the set $\mathcal{U} = Pow(\Sigma^*)$, and the sets:

$$\{A \in \mathcal{U} : \mathbf{P}^A = \mathbf{NP}^A\}$$

$$\{B \in \mathcal{U} : \mathbf{P}^B \neq \mathbf{NP}^B\}$$

- Can we compare these two sets, and find which is *larger*?

Theorem (Bennet, Gill)

$$\mathbf{Pr}_{B\subseteq\Sigma^*}\left[\mathbf{P}^B \neq \mathbf{NP}^B\right] = 1$$

See H. Vollmer & K.W. Wagner, "Measure one Results in Computational Complexity Theory"

The Polynomial Hierarchy

Polynomial Hierarchy Definition

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathbf{P}$
- $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
- $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
- $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $$\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$$

The Polynomial Hierarchy

Polynomial Hierarchy Definition

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathbf{P}$
- $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$
- $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$
- $\Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$
- $$\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$$

- $\Sigma_0^p = \mathbf{P}$
- $\Delta_1^p = \mathbf{P}$, $\Sigma_1^p = \mathbf{NP}$, $\Pi_1^p = co\mathbf{NP}$
- $\Delta_2^p = \mathbf{P}^{\mathbf{NP}}$, $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$, $\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}}$

The Polynomial Hierarchy

$$\begin{tikzcd}
\vdots & & \vdots \\
& \Delta_3^p = \mathbf{P}^{\mathbf{NP}^{\mathbf{NP}}} \arrow[ul] \arrow[ur] & \\
\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}} \arrow[ur] & & \Sigma_2^p = \mathbf{NP}^{\mathbf{NP}} \arrow[ul] \\
& \Delta_2^p = \mathbf{P}^{\mathbf{NP}} \arrow[ul] \arrow[ur] & \\
\Pi_1^p = co\mathbf{NP} \arrow[ur] & & \Sigma_1^p = \mathbf{NP} \arrow[ul] \\
& \Delta_0^p = \Sigma_0^p = \Pi_0^p = \Delta_1^p = \mathbf{P} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

- $\Sigma_i^p, \Pi_i^p \subseteq \Sigma_{i+1}^p$
- $A, B \in \Sigma_i^p \Rightarrow$ $A \cup B \in \Sigma_i^p$, $A \cap B \in \Sigma_i^p$
- $A \in \Pi_i^p \Rightarrow$ $\overline{A} \in \Sigma_i^p$
- $A, B \in \Delta_i^p \Rightarrow$ $A \cup B$, $A \cap B$ and $\overline{A} \in \Delta_i^p$

Theorem

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Theorem

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Proof (by Induction):

Th.17.8 (p.425) in [1]

- $\boxed{\text{For } i = 1:}$
  $\{x; y : (x, y) \in R\} \in \mathbf{P}$,so $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP}$ ✓

Theorem

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Proof (by Induction):

Th.17.8 (p.425) in [1]

- $\boxed{\text{For } i = 1\text{:}}$
  $\{x; y : (x, y) \in R\} \in \mathbf{P}$,so $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP}$ ✓

- $\boxed{\text{For } i > 1\text{:}}$
  If $\exists R \in \Pi_{i-1}^p$, we must show that $L \in \Sigma_i^p \Rightarrow$
  $\exists$ NTM with Σ_{i-1}^p oracle: NTM(x) guesses a y and asks Π_{i-1}^p oracle whether $(x, y) \notin R$.

Proof (*cont'd*):
If $L \in \Sigma_i^p$, we must show the existence or R:

- $L \in \Sigma_i^p \Rightarrow \exists$ NTM M^K, $K \in \Sigma_{i-1}^p$, which decides L.
- $K \in \Sigma_{i-1}^p \Rightarrow \exists S \in \Pi_{i-2}^p : (z \in K \Leftrightarrow \exists w : (z, w) \in S)$.
- We must describe a relation R (we know: $x \in L \Leftrightarrow$ accepting computation of $M^K(x)$)
- Query Steps: "yes"$\rightarrow z_i$ has a certificate w_i st $(z_i, w_i) \in S$.
- So, $R(x) =$"$(x, y) \in R$ *iff yrecords an accepting computation of* $M^?$ *on* x *, together with a certificate* w_i *for each* **yes** *query* z_i *in the computation.*"
- We must show $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$:
    - Check that all steps of $M^?$ are legal (*poly time*).
    - Check that $(z_i, w_i) \in S$ (*in* Π_{i-2}^p*, and thus in* Π_{i-1}^p).
    - For all "no" queries z_i', check $z_i' \notin K$ (*another* Π_{i-1}^p). □

Corollary

Let L be a language , and $i \geq 1$. $L \in \Pi_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Σ_{i-1}^p and

$$L = \{x : \forall y, |y| \leq |x|^k, s.t. : (x, y) \in R\}$$

Corollary

Let L be a language , and $i \geq 1$. $L \in \Pi_i^p$ iff there is a polynomially balanced relation R such that the language $\{x; y : (x, y) \in R\}$ is in Σ_{i-1}^p and

$$L = \{x : \forall y, |y| \leq |x|^k, s.t. : (x, y) \in R\}$$

Corollary

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced, polynomially-time decicable $(i+1)$-ary relation R such that:

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 ... Q y_i, s.t. : (x, y_1, ..., y_i) \in R\}$$

where the i^{th} quantifier Q is $\forall$, if i is even, and $\exists$, if i is odd.

Remark

$$\Sigma_i^p = (\underbrace{\exists\forall\exists\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\forall\exists\forall\cdots Q_i'}_{i \text{ quantifiers}}) \qquad \Pi_i^p = (\underbrace{\forall\exists\forall\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\exists\forall\exists\cdots Q_i'}_{i \text{ quantifiers}})$$

Remark

$\Sigma_i^p = (\underbrace{\exists\forall\exists\cdots Q_i}_{i\text{ quantifiers}} / \underbrace{\forall\exists\forall\cdots Q_i'}_{i\text{ quantifiers}})$ $\quad\quad$ $\Pi_i^p = (\underbrace{\forall\exists\forall\cdots Q_i}_{i\text{ quantifiers}} / \underbrace{\exists\forall\exists\cdots Q_i'}_{i\text{ quantifiers}})$

Theorem

If for some $i \geq 1$, $\Sigma_i^p = \Pi_i^p$, then for all $j > i$:

$$\Sigma_j^p = \Pi_j^p = \Delta_j^p = \Sigma_i^p$$

Or, the polynomial hierarchy *collapses* to the i^{th} level.

Remark

$\Sigma_i^p = (\underbrace{\exists\forall\exists\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\forall\exists\forall\cdots Q_i'}_{i \text{ quantifiers}})$ $\qquad$ $\Pi_i^p = (\underbrace{\forall\exists\forall\cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\exists\forall\exists\cdots Q_i'}_{i \text{ quantifiers}})$

Theorem

If for some $i \geq 1$, $\Sigma_i^p = \Pi_i^p$, then for all $j > i$:

$$\Sigma_j^p = \Pi_j^p = \Delta_j^p = \Sigma_i^p$$

Or, the polynomial hierarchy *collapses* to the i^{th} level.

Proof: Th.17.9 (p.427) in [1]

- It suffices to show that: $\Sigma_i^p = \Pi_i^p \Rightarrow \Sigma_{i+1}^p = \Sigma_i^p$.
- Let $L \in \Sigma_{i+1}^p \Rightarrow \exists R \in \Pi_i^p$: $L = \{x | \exists y : (x, y) \in R\}$
- $\Pi_i^p = \Sigma_i^p \Rightarrow R \in \Sigma_i^p$
- $(x, y) \in R \Leftrightarrow \exists z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$.
- So, $x \in L \Leftrightarrow \exists y; z : (x, y, z) \in S$, $S \in \Pi_{i-1}^p$, hence $L \in \Sigma_i^p$. $\square$

Corollary

If **P**=**NP**, or even **NP**=co**NP**, the Polynomial Hierarchy collapses to the first level.

Corollary

If **P**=**NP**, or even **NP**=co**NP**, the Polynomial Hierarchy collapses to the first level.

QSAT$_i$ Definition

Given expression ϕ, with Boolean variables partitioned into i sets X_i,is ϕ satisfied by the overall truth assignment of the expression:

$$\exists X_1 \forall X_2 \exists X_3 .....QX_i \phi$$

where Q is $\exists$ if i is *odd*, and $\forall$ if i is even.

Theorem

For all $i \geq 1$ QSAT$_i$ is Σ_i^p-complete.

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof: Th.17.11 (p.429) in [1]

- Let L is **PH**-complete.
- Since $L \in \mathbf{PH}$, $\exists i \geq 0 : L \in \Sigma_i^p$.
- But any $L' \in \Sigma_{i+1}^p$ reduces to L.
- Since PH is closed under reductions, we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$. $\square$

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof: Th.17.11 (p.429) in [1]

- Let L is **PH**-complete.
- Since $L \in \mathbf{PH}$, $\exists i \geq 0 : L \in \Sigma_i^p$.
- But any $L' \in \Sigma_{i+1}^p$ reduces to L.
- Since PH is closed under reductions, we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$. □

Theorem

$\mathbf{PH} \subseteq \mathbf{PSPACE}$

- $\mathbf{PH} \stackrel{?}{=} \mathbf{PSPACE}$ (**Open**). If it was, then **PH** has complete problems, so it collapses to some finite level.

Relativized Results

Let's see how the inclusion of the Polynomial Hierarchy to Polynomial Space, and the inclusions of each level of **PH** to the next relativizes:

- $\mathbf{PH}^A \neq \mathbf{PSPACE}^A$ relative to *some* oracle $A \subseteq \Sigma^*$.
  (Yao 1985, Håstad 1986)
- $\mathbf{Pr}_A[\mathbf{PH}^A \neq \mathbf{PSPACE}^A] = 1$
  (Cai 1986, Babai 1987)
- $(\forall i \in \mathbb{N})\ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}$ relative to *some* oracle $A \subseteq \Sigma^*$.
  (Yao 1985, Håstad 1986)
- $\mathbf{Pr}_A[(\forall i \in \mathbb{N})\ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}] = 1$
  (Rossman-Servedio-Tan, 2015)

Self-Reducibility of SAT

- For a Boolean formula ϕ with n variables and m clauses.
- It is easy to see that:

$$\boxed{\phi \in \mathrm{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \mathrm{SAT}) \vee (\phi|_{x_1=1} \in \mathrm{SAT})}$$

- Thus, we can **self-reduce** SAT to instances of smaller size.
- Self-Reducibility Tree of depth n:

Self-Reducibility of SAT

- For a Boolean formula ϕ with n variables and m clauses.
- It is easy to see that:

$$\boxed{\phi \in \mathrm{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \mathrm{SAT}) \vee (\phi|_{x_1=1} \in \mathrm{SAT})}$$

- Thus, we can **self-reduce** SAT to instances of smaller size.
- Self-Reducibility Tree of depth n:

Example

$$\begin{tikzcd}
 & & & \phi(x_1, x_2) \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & \phi|_{x_1=0} \arrow[dl, dash] \arrow[dr, dash] & & & & \phi|_{x_1=1} \arrow[dl, dash] \arrow[dr, dash] & \\
\phi|_{x_1=0,x_2=0} & & \phi|_{x_1=0,x_2=1} & & \phi|_{x_1=1,x_2=0} & & \phi|_{x_1=1,x_2=1}
\end{tikzcd}$$

Self-Reducibility of SAT

Definition (FSAT)

FSAT: Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying truth assignment for ϕ. Otherwise return “no”.

Self-Reducibility of SAT

Definition (FSAT)

FSAT: Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying truth assignment for ϕ. Otherwise return “no”.

- **FP** is the function analogue of **P**: it contains functions computable by a DTM in poly-time.
- FSAT $\in$ **FP** $\Rightarrow$ SAT $\in$ **P**.
- What about the opposite?

Self-Reducibility of SAT

Definition (FSAT)

FSAT: Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying truth assignment for ϕ. Otherwise return “no”.

- **FP** is the function analogue of **P**: it contains functions computable by a DTM in poly-time.
- FSAT $\in$ **FP** $\Rightarrow$ SAT $\in$ **P**.
- What about the opposite?
- If SAT $\in$ **P**, we can use the self-reducibility property to fix variables one-by-one, and retrieve a solution.
- We only need $2n$ calls to the *alleged* poly-time algorithm for SAT.

What about TSP?

- We can solve TSP using a hypothetical algorithm for the **NP**-complete decision version of TSP:

What about TSP?

- We can solve TSP using a hypothetical algorithm for the **NP**-complete decision version of TSP:
- We can find the cost of the optimum tour by **binary search** (in the interval $[0, 2^n]$).
- When we find the optimum cost C, we fix it, and start changing intercity distances one-by one, by setting each distance to $C + 1$.
- We then ask the **NP**-oracle if there still is a tour of optimum cost at most C:
  - If there is, then this edge is not in the optimum tour.
  - If there is not, we know that this edge is in the optimum tour.
- After at most n^2 (polynomial) oracle queries, we can extract the optimum tour, and thus have the solution to TSP.

The Classes $\mathbf{P}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}$

- $\mathbf{P}^{\mathrm{SAT}}$ is the class of languages decided in pol time with a SAT oracle (*Polynomial number of adaptive queries*).
- SAT is **NP**-complete $\Rightarrow \mathbf{P}^{\mathrm{SAT}} = \mathbf{P}^{\mathbf{NP}}$.
- $\mathbf{FP}^{\mathbf{NP}}$ is the class of **functions** that can be computed by a poly-time DTM with a SAT oracle.
- FSAT, TSP $\in \mathbf{FP}^{\mathbf{NP}}$.

The Classes $\mathbf{P}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}$

- $\mathbf{P}^{\mathrm{SAT}}$ is the class of languages decided in pol time with a SAT oracle (*Polynomial number of adaptive queries*).
- SAT is **NP**-complete $\Rightarrow \mathbf{P}^{\mathrm{SAT}} = \mathbf{P}^{\mathbf{NP}}$.
- $\mathbf{FP}^{\mathbf{NP}}$ is the class of **functions** that can be computed by a poly-time DTM with a SAT oracle.
- FSAT, TSP $\in \mathbf{FP}^{\mathbf{NP}}$.

Definition (Reductions for Function Problems)

A function problem A reduces to B if there exists $R, S \in \mathbf{FL}$ such that:

- $x \in A \Rightarrow R(x) \in B$.
- If z is a correct output of $R(x)$, then $S(z)$ is a correct output of x.

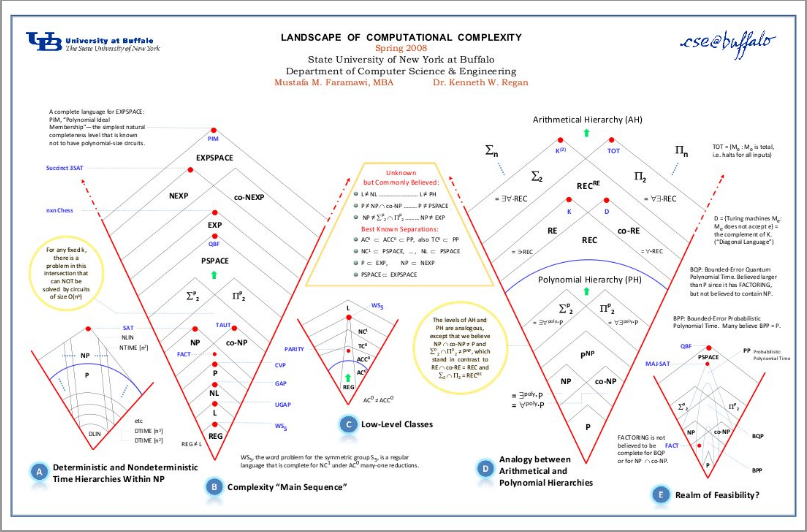
Theorem

TSP *is* $\mathbf{FP}^{\mathbf{NP}}$*-complete.*

Summary

- Oracle TMs have one-step oracle access to some language.
- There exist oracles $A, B \subseteq \Sigma^*$ such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$.
- Relativizing results hold for *every* oracle.
- A Cook reduction $A \leq_T^p B$ is a poly-time TM deciding A, by using B as an oracle.
- The Polynomial Hierarchy can be viewed as:
  - Oracle hierarchy of consecutive **NP** oracles.
  - Quantifier hierarchy of alternating quantifiers.
- If for some $i \geq 1$ $\Sigma_i^p = \Pi_i^p$, or there is a **PH**-complete problem, then PH collapses to some finite level.
- Optimization problems with decision version in **NP** (such as TSP) are in $\mathbf{FP}^{\mathbf{NP}}$.

The Complexity Universe



University at Buffalo
The State University of New York

LANDSCAPE OF COMPUTATIONAL COMPLEXITY
Spring 2008
State University of New York at Buffalo
Department of Computer Science & Engineering
Mustafa M. Faramawi, MBA   Dr. Kenneth W. Regan

cse@buffalo

A complete language for EXPSPACE: PIM, "Polynomial Ideal Membership"— the simplest natural completeness level that is known not to have polynomial-size circuits.

For any fixed k, there is a problem in this intersection that can NOT be solved by circuits of size $O(n^k)$

Unknown but Commonly Believed:

- $L \neq NL$ .... $L \neq PH$
- $P \neq NP \cap co\text{-}NP$ .... $P \neq PSPACE$
- $NP \neq \Sigma^p_2 \cap \Pi^p_2$ .... $NP \neq EXP$

Best Known Separations:

- $AC^0 \subset ACC^0 \subset PP$, also $TC^0 \subset PP$
- $NC^1 \subset PSPACE$, ..., $NL \subset PSPACE$
- $P \subset EXP$, $NP \subset NEXP$
- $PSPACE \subset EXPSPACE$

The levels of AH and PH are analogous, except that we believe $NP \cap co\text{-}NP \neq P$ and $\Sigma^p_2 \cap \Pi^p_2 \neq P^{NP}$, which stand in contrast to $RE \cap co\text{-}RE = REC$ and $\Sigma_2 \cap \Pi_2 = REC^{RE}$

WS_5, the word problem for the symmetric group S_5, is a regular language that is complete for NC^1 under AC^0 many-one reductions.

TOT = {M_e : M_e is total, i.e. halts for all inputs}

D = {Turing machines M_e: M_e does not accept e} = the complement of K. ("Diagonal Language")

BQP: Bounded-Error Quantum Polynomial Time. Believed larger than P since it has FACTORING, but not believed to contain NP.

BPP: Bounded-Error Probabilistic Polynomial Time. Many believe BPP = P.

FACTORING is not believed to be complete for BQP or for NP ∩ co-NP.

A Deterministic and Nondeterministic Time Hierarchies Within NP

B Complexity "Main Sequence"

C Low-Level Classes

D Analogy between Arithmetical and Polynomial Hierarchies

E Realm of Feasibility?

Contents

Problems...

- After years of efforts, there are problems in **NP** without a polynomial-time algorithm or a completeness proof.
- Famous examples: FACTORING$_D$, GI (Graph Isomorphism). (where FACTORING$_D$ is the problem of *deciding* if a given number has a factor $\leq k$).
- So, are there **NP** problems that are neither in **P** nor **NP**-complete?

Degrees

- The **$\leq_T^p$-degree** of a language A consists of all languages L such that $L \equiv_T^p A$ (that is, $L \leq_T^p A \ \wedge \ A \leq_T^p L$).

Degrees

- The **$\leq_T^p$-degree** of a language A consists of all languages L such that $L \equiv_T^p A$ (that is, $L \leq_T^p A \ \wedge\ A \leq_T^p L$).
- There are three possibilities:
  - $\mathbf{P} = \mathbf{NP}$, thus all languages in **NP** are $\leq_T^p$-complete for **NP**, so **NP** contains *exactly one* $\leq_T^p$-degree.
  - $\mathbf{P} \neq \mathbf{NP}$, and **NP** contains *two different* degrees: **P** and **NP**-complete languages.
  - $\mathbf{P} \neq \mathbf{NP}$, and **NP** contains more degrees, so there exists a language in $\mathbf{NP} \setminus \mathbf{P}$ that is not **NP**-complete.

Degrees

- The **$\leq_T^p$-degree** of a language A consists of all languages L such that $L \equiv_T^p A$ (that is, $L \leq_T^p A \ \wedge \ A \leq_T^p L$).
- There are three possibilities:
  - $\mathbf{P} = \mathbf{NP}$, thus all languages in **NP** are $\leq_T^p$-complete for **NP**, so **NP** contains *exactly one* $\leq_T^p$-degree.
  - $\mathbf{P} \neq \mathbf{NP}$, and **NP** contains *two different* degrees: **P** and **NP**-complete languages.
  - $\mathbf{P} \neq \mathbf{NP}$, and **NP** contains more degrees, so there exists a language in $\mathbf{NP} \setminus \mathbf{P}$ that is not **NP**-complete.
- We will show that the second case cannot happen.

Enumerations

- Recall that any string can potentially encode a TM.
(*We map all the invalid encodings to the "empty" TM M_0, which reject all strings.*)
- A TM M is encoded by infinitely many strings.
- So, there exists a function $e(x)$ such that:
  1. For every $x \in \Sigma^*$, $e(x)$ represents a TM.
  2. Every TM is represented by at least one $e(x)$.
  3. The code of the TM $e(x)$ can be easily decoded.
- Such a function is called an **enumeration** of TMs (Deterministic or Nondeterministic).

Enumerations

- When we consider classes like **P** or **NP**, we can easily enumerate only these machines, a *subclass* of all DTMs (NTMs):

Enumerations

- When we consider classes like **P** or **NP**, we can easily enumerate only these machines, a *subclass* of all DTMs (NTMs):
- Recall that if a function is **time-constructible**, there exists a DTM halting after exactly $t(n)$ moves. Such a machine is called a $t(n)$**-clock machine**.
- For any DTM M_1, we can attach a $t(n)$-clock machine M_2 and obtain a “product” machine $M_3 = \langle M_1, M_2 \rangle$, which halts if either M_1 or M_2 halts, and accepts only if M_1 accepts.

Enumerations

- Consider the functions $p_i(n) = n^i, i \geq 1$.
- If $\{M_x\}$ is an enumeration of DTMs, let $M_{\langle x,i \rangle}$ be the machine M_x attached with a $p_i(n)$-clock machine.
- Then, $\{M_{\langle x,i \rangle}\}$ is an **enumeration of all polynomial-time clocked machines**, and it is an enumeration of languages in **P**, such that:
    - Every machine $M_{\langle x,i \rangle}$ accepts a language in **P**.
    - Every language in **P** is accepted by at least a machine in the enumeration (in fact, by infinite number of machines).

Enumerations

- The same holds for **NP**.
  (*enumerate all poly-time alarm clocked NTMs*)
- We can do the same trick with **space**, using a **yardstick**, a DTM that halts after visiting *exactly* $s(n)$ memory cells.
- We can also enumerate all the functions in **FP**, and all polynomial-time *oracle* DTMs or NTMs.

Enumerations

- The same holds for **NP**.
  (*enumerate all poly-time alarm clocked NTMs*)
- We can do the same trick with **space**, using a **yardstick**, a DTM that halts after visiting *exactly* $s(n)$ memory cells.
- We can also enumerate all the functions in **FP**, and all polynomial-time *oracle* DTMs or NTMs.
- This list will **not** contain *all* the poly-time bounded machines!
  (Reminder: It is undecidable to determine whether a given TM halts in polynomial time for all inputs.)

Ladner’s Theorem

Theorem (Ladner)

If **P** $\neq$ **NP**, *there exists a language in* **NP**, *which is neither in* **P** *nor* **NP**-*complete.*

Ladner’s Theorem

Theorem (Ladner)

If **P** $\neq$ **NP**, *there exists a language in* **NP**, *which is neither in* **P** *nor* **NP**-*complete.*

Proof (*Blowing holes in* SAT): Th. 14.1 (p.330) in [1]

- Idea: We will construct a language A by taking an **NP**-complete language, and “blow holes” to it, so that it is no longer **NP**-complete, neither in **P**.
- Let $\{M_i\}$ an enumeration of all polynomial-time *clocked* TMs.
- Let $\{F_i\}$ an enumeration of all polynomial-time *clocked* functions.

Ladner’s Theorem

Theorem (Ladner)

If **P** $\neq$ **NP**, *there exists a language in* **NP**, *which is neither in* **P** *nor* **NP**-*complete.*

Proof (*Blowing holes in* SAT): Th. 14.1 (p.330) in [1]

- Idea: We will construct a language A by taking an **NP**-complete language, and “blow holes” to it, so that it is no longer **NP**-complete, neither in **P**.
- Let $\{M_i\}$ an enumeration of all polynomial-time *clocked* TMs.
- Let $\{F_i\}$ an enumeration of all polynomial-time *clocked* functions.
- Define A as follows:

$$A = \{x \mid x \in \text{SAT} \ \wedge \ f(|x|) \text{ is even}\}$$

Ladner’s Theorem

Proof (*cont’d*):

- If $f \in \mathbf{FP}$, then $A \in \mathbf{NP}$: Guess a truth assignment, compute $f(|x|)$ and verify.
- We define f by a polynomial-time TM M_f computing it.
- Let also M_{SAT} be the machine that decides SAT, and $f(0) = f(1) = 2$.

Ladner's Theorem

Proof (*cont'd*):

- If $f \in \mathbf{FP}$, then $A \in \mathbf{NP}$: Guess a truth assignment, compute $f(|x|)$ and verify.
- We define f by a polynomial-time TM M_f computing it.
- Let also M_{SAT} be the machine that decides SAT, and $f(0) = f(1) = 2$.
- On input 1^n, M_f operates in two stages, each lasting for exactly n steps:
- $\boxed{\text{First Stage}}$
  M_f computes $f(0), f(1), \ldots$ until it runs out of time.
  - Let $f(x) = k$ the last value of f it was able to compute.
  - Then M_f outputs either k or $k+1$, to be determined in the next stage:

Ladner’s Theorem

Proof (*cont’d*):

- $\boxed{\text{Second Stage}}$

  If $k = 2i$:
  - M_f tries to find a $z \in \{0,1\}^*$ such that $M_i(z)$ outputs the *wrong* answer to “$z \in A$” question ($M_i(z) \neq A(z)$):
    - Simulate $M_i(z), M_{\text{SAT}}(z), f(|z|)$ for all z in lexicographic order.
    - If such a string is found in the allotted time, output $k+1$, else output k.

  If $k = 2i - 1$:
  - M_f tries to find a string z such that $F_i(z)$ is an *incorrect* Karp reduction from SAT to A ($M_{\text{SAT}}(z) \neq A(F_i(z))$):
    - Simulate $F_i(z), M_{\text{SAT}}(z), M_{\text{SAT}}(F_i(z)), f(|F_i(z)|)$ for all z in lexicographic order.
    - If such a string is found in the allotted time, output $k+1$, else output k.
- M_f runs in polynomial time.
- $f(n+1) \geq f(n)$.

Ladner’s Theorem

Proof (*cont’d*):

- **We claim that $A \notin$ P:**

Ladner’s Theorem

Proof (*cont’d*):

- **We claim that $A \notin \mathbf{P}$:**
- Suppose that $A \in \mathbf{P}$. Then, there is an i s.t. $L(M_i) = A$.
- Then, the second stage of M_f with $k = 2i$ will never find a z satisfying the desired property.
- $f(n) = 2i$ for all $n \geq n_0$, for some n_0.
- So, $f(n)$ is even for all but finitely many n.
- A coincides with SAT on all but finitely many input sizes.
- Then SAT $\in \mathbf{P}$, contradiction!

Ladner’s Theorem

Proof (*cont’d*):

- **We claim that A is not NP-complete:**

Ladner’s Theorem

Proof (*cont’d*):

- **We claim that *A* is not NP-complete:**
- Suppose that A is **NP**-complete, then there is a reduction F_i from SAT to A.
- Then, the second stage of M_f with $k = 2i - 1$ will never find a z satisfying the desired property.
- So, $f(n)$ is odd on all but finitely many input sizes.
- Then A is a finite language, hence in **P**, contradiction! $\square$

- Using the same technique, we can prove an analog of *Post’s problem* in Recursion Theory:

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, there exist* $A, B \in \mathbf{NP}$ *such that* $A \not\leq_T^p B$ *and* $B \not\leq_T^p A$.

- Using the same technique, we can prove an analog of *Post’s problem* in Recursion Theory:

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, there exist* $A, B \in \mathbf{NP}$ *such that* $A \not\leq_T^p B$ *and* $B \not\leq_T^p A$.

- Ladner’s Theorem (*generalized by Schöning*) implies also that:

Corollary

If $\mathbf{P} \neq \mathbf{NP}$*, then for every language* $B \in \mathbf{NP} \setminus \mathbf{P}$*, there exists a set* $A \in \mathbf{NP} \setminus \mathbf{P}$ *such that* $A \leq_T^p B$ *and* $B \not\leq_T^p A$.

- Using the same technique, we can prove an analog of *Post’s problem* in Recursion Theory:

Theorem

If $\mathbf{P} \neq \mathbf{NP}$, *there exist* $A, B \in \mathbf{NP}$ *such that* $A \not\leq_T^p B$ *and* $B \not\leq_T^p A$.

- Ladner’s Theorem (*generalized by Schöning*) implies also that:

Corollary

If $\mathbf{P} \neq \mathbf{NP}$, *then for every language* $B \in \mathbf{NP} \setminus \mathbf{P}$, *there exists a set* $A \in \mathbf{NP} \setminus \mathbf{P}$ *such that* $A \leq_T^p B$ *and* $B \not\leq_T^p A$.

So, if $\mathbf{P} \neq \mathbf{NP}$, then $\mathbf{NP}$ contains *infinitely many* distinct $\leq_T^p$-degrees.

Polynomial-Time Isomorphism

- All **NP**-complete problems are related through reductions.
- Many reductions can be converted to stronger relations:

Definition

Two languages $A, B \subseteq \Sigma^*$ are *polynomial-time isomorphic* if there exists a function $h : \Sigma^* \to \Sigma^*$ such that:

1. h is a bijection.
2. For all $x \in \Sigma^*$: $x \in A \Leftrightarrow h(x) \in B$.
3. Both h and h^{-1} are polynomial-time computable.

Functions h and h^{-1} are then called *polynomial-time isomorphisms*.

- Which reductions are polynomial-time isomorphisms?

Padding Functions

Definition

Let $L \subseteq \Sigma^*$ be a language. We say that function $pad : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a *padding function* for L if it has the following properties:

1. It is computable in logarithmic space.
2. Forall $x, y \in \Sigma^*$, $pad(x, y) \in L \Leftrightarrow x \in L$.
3. Forall $x, y \in \Sigma^*$, $|pad(x, y)| > |x| + |y|$
4. There is a logarithmic-space algorithm, which, given $pad(x, y)$ recovers y.

- Such languages are called *paddable*.
- Function *pad* is essentially a length-increasing reduction from L to itself that “encodes” another string y into the instance of L.

Padding Functions Examples

Example (SAT)

Let x an instance with n variables and m clauses. Let $y \in \Sigma^*$:
$pad(x, y)$ is an instance of SAT containing all clauses of x, plus $m + |y|$ more clauses, and $|y| + 1$ more variables.

- The first m clauses are copies of x_{n+1} clause.
- The last $m + i^{th}$ $(i = 1, \cdots, |y|)$ are either $\neg x_{n+i+1}$ (if $y(i) = 0$) or x_{n+i+1} (if $y(i) = 1$).

Is that a padding function?

1. It is log-space computable.
2. It doesn't affect x's satisfiability.
3. It is length increasing.
4. Given $pad(x, y)$ we can find where the "added" part begins.

Padding Functions

- We would like to have this kind of implication:
  $(A \leq_m^p B) \wedge (B \leq_m^p A) \overset{?}{\Rightarrow} (A \text{ isomorphic to } B)$.
- But, unfortunately, this is **not** sufficient.
- We finally want to have a polynomial-time version of Schröder-Bernstein Theorem:

Padding Functions

- We would like to have this kind of implication: $(A \leq_m^p B) \wedge (B \leq_m^p A) \overset{?}{\Rightarrow} (A \text{ isomorphic to } B)$.
- But, unfortunately, this is **not** sufficient.
- We finally want to have a polynomial-time version of Schröder-Bernstein Theorem:

Theorem (Schröder-Bernstein)

If there exists a 1-1 mapping from a set A to a set B, and a 1-1 mapping from B to A, then there is a bijection between A and B.

- To achieve this analogy, we need to “enhance” our reductions with the previous features (1-1, length increasing, and polynomial time computable and invertible).

Padding Functions

- We can use padding function to transform regular reductions to "desired" ones:

Theorem

Let R be a reduction from A to B, and pad a padding function for B. Then, the function mapping $x \in \Sigma^*$ *to* $pad(R(x), x)$ *is a length-increasing 1-1 reduction. Furthermore, there exists* R^{-1}*, computable in logarithmic space, which given* $pad(R(x), x)$ *recovers* x*.*

Padding Functions

- We can use padding function to transform regular reductions to “desired” ones:

Theorem

Let R be a reduction from A to B, and pad a padding function for B. Then, the function mapping $x \in \Sigma^*$ to $pad(R(x), x)$ is a length-increasing 1-1 reduction. Furthermore, there exists R^{-1}, computable in logarithmic space, which given $pad(R(x), x)$ recovers x.

Theorem (Polynomial-time version of Schröder-Bernstein Theorem)

Let A and B be paddable languages. If $A \leq_m^p B$ and $B \leq_m^p A$, then A and B are polynomial-time isomorphic.

Padding Functions

Corollary

The following ***NP****-complete languages are pol. isomorphic:*
SAT, VERTEX COVER, HAMILTON PATH, CLIQUE, MAX CUT, TRIPARTITE MATCHING, KNAPSACK

Padding Functions

Corollary

The following ***NP****-complete languages are pol. isomorphic:*
SAT, VERTEX COVER, HAMILTON PATH, CLIQUE, MAX CUT, TRIPARTITE MATCHING, KNAPSACK

- We can (almost trivially) find padding functions for every known **NP**-complete problem.

Definition (Berman-Hartmanis Conjecture)

All **NP**-complete languages are polynomial-time isomorphic to each other!

Padding Functions

Corollary

The following **NP***-complete languages are pol. isomorphic:*
SAT, VERTEX COVER, HAMILTON PATH, CLIQUE, MAX CUT, TRIPARTITE MATCHING, KNAPSACK

- We can (almost trivially) find padding functions for every known **NP**-complete problem.

Definition (Berman-Hartmanis Conjecture)

All **NP**-complete languages are polynomial-time isomorphic to each other!

- Berman-Hartmanis Conjecture $\Rightarrow$ $\mathbf{P} \neq \mathbf{NP}$ (*why?*)

Translation Results

Theorem

If $\mathbf{NEXP} \neq \mathbf{EXP}$, *then* $\mathbf{P} \neq \mathbf{NP}$.

Translation Results

Theorem

If **NEXP** $\neq$ **EXP**, *then* **P** $\neq$ **NP**.

Proof:

- We will prove that if **P** $=$ **NP**, then **NEXP** $=$ **EXP**.
- Let $L \in$ **NTIME**$[2^{n^c}]$ and M a TM deciding it. We define:

$$L_p = \{x\$^{2^{|x|^c}} \mid x \in L\}$$

Translation Results

Theorem

If $\mathbf{NEXP} \neq \mathbf{EXP}$, *then* $\mathbf{P} \neq \mathbf{NP}$.

Proof:

- We will prove that if $\mathbf{P} = \mathbf{NP}$, then $\mathbf{NEXP} = \mathbf{EXP}$.
- Let $L \in \mathbf{NTIME}[2^{n^c}]$ and M a TM deciding it. We define:

$$L_p = \{x\$^{2^{|x|^c}} \mid x \in L\}$$

- L_p is in $\mathbf{NP}$: Simulate $M(x)$ for $2^{|x|^c}$ steps and output the answer. The running time of this machine is polynomial in its input size.
- By our assumption, $L_p \in \mathbf{P}$.
- We can use the machine in $\mathbf{P}$ to decide L in $\mathbf{EXP}$: on input x, pad it using $2^{|x|^c}$ \$'s, and use the machine in $\mathbf{P}$ to decide L_p.
- The running time is $2^{|x|^c}$, so $L \in \mathbf{EXP}$. $\square$

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Theorem

$$\mathbf{E} \neq \mathbf{PSPACE}$$

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Theorem

$$\mathbf{E} \neq \mathbf{PSPACE}$$

Proof:

- Assume that $\mathbf{E} = \mathbf{PSPACE}$.
- Let $L \in \mathbf{DTIME}[2^{n^2}]$.

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Theorem

$$\mathbf{E} \neq \mathbf{PSPACE}$$

Proof:

- Assume that $\mathbf{E} = \mathbf{PSPACE}$.
- Let $L \in \mathbf{DTIME}[2^{n^2}]$.
- We define:

$$L_p = \{x\$^\ell \mid x \in L \wedge |x\$^\ell| = |x|^2\}$$

- $L_p \in \mathbf{DTIME}[2^n]$
- From our assumption: $L_p \in \mathbf{PSPACE} \Rightarrow L_p \in \mathbf{DSPACE}[n^k]$, for some $k \in \mathbb{N}$.

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Theorem

$$\mathbf{E} \neq \mathbf{PSPACE}$$

Proof (*cont'd*):

- We can convert this n^k-space-bounded machine to another, deciding L:
- Given x, add $\ell = |x|^2 - |x|$ \$'s, and simulate the n^k-space-bounded machine on the padded input.
- We used $|x|^{2k}$ space, so $L \in \mathbf{PSPACE} \Rightarrow \mathbf{DTIME}[2^{n^2}] \subseteq \mathbf{PSPACE}$.

Separation Results

- Let $\mathbf{E} = \mathbf{DTIME}[2^{\mathcal{O}(n)}]$.

Theorem

$$\mathbf{E} \neq \mathbf{PSPACE}$$

Proof (*cont'd*):

- We can convert this n^k-space-bounded machine to another, deciding L:
- Given x, add $\ell = |x|^2 - |x|$ \$'s, and simulate the n^k-space-bounded machine on the padded input.
- We used $|x|^{2k}$ space, so $L \in \mathbf{PSPACE} \Rightarrow \mathbf{DTIME}[2^{n^2}] \subseteq \mathbf{PSPACE}$.
- But, $\mathbf{E} \subsetneq \mathbf{DTIME}[2^{n^2}]$, and so $\mathbf{E} \neq \mathbf{PSPACE}$. $\square$

Density of Languages

Definition

Let $L \subseteq \Sigma^*$ be a language. We define as its **density** the following function from $\mathbb{N} \rightarrow \mathbb{N}$:

$$dens_L(n) = |\{x \in L : |x| \leq n\}|$$

- $dens_L(n)$ is the *number of strings* in L of length up to n.

Density of Languages

Definition

Let $L \subseteq \Sigma^*$ be a language. We define as its **density** the following function from $\mathbb{N} \to \mathbb{N}$:

$$dens_L(n) = |\{x \in L : |x| \leq n\}|$$

- $dens_L(n)$ is the *number of strings* in L of length up to n.

Theorem

If $A, B \subseteq \Sigma^*$ are polynomial-time isomorphic, then $dens_A$ and $dens_B$ are polynomially related.

Proof:

- All $x \in A$ with $|x| \leq n$ are mapped to $y \in B$ with $|y| \leq p(n)$, where p is the polynomial bound of the isomorphism.
- The mapping is 1-1, so $dens_A(n) \leq dens_B(p(n))$. □

Sparse Languages

Definition

A language L is *sparse* if there exists a polynomial q such that for every $n \in \mathbb{N} : dens_L(n) \leq q(n)$.

Sparse Languages

Definition

A language L is *sparse* if there exists a polynomial q such that for every $n \in \mathbb{N} : dens_L(n) \leq q(n)$.

Theorem

If a language A is paddable, then it is not sparse.

Sparse Languages

Definition

A language L is *sparse* if there exists a polynomial q such that for every $n \in \mathbb{N} : dens_L(n) \leq q(n)$.

Theorem

If a language A is paddable, then it is not sparse.

Proof:

- Let $A \subseteq \Sigma^*$ with padding function $p : \Sigma^* \times \Sigma^* \to \Sigma^*$.
- Suppose that A is sparse: $\exists q \, \forall n \in \mathbb{N} : dens_A(n) \leq q(n)$.
- Since $p \in \mathbf{FP}$, $\exists \, r \in poly(n) : |p(x, y)| \leq r(|x| + |y|)$.

Sparse Languages

Definition

A language L is *sparse* if there exists a polynomial q such that for every $n \in \mathbb{N} : dens_L(n) \leq q(n)$.

Theorem

If a language A is paddable, then it is not sparse.

Proof:

- Let $A \subseteq \Sigma^*$ with padding function $p : \Sigma^* \times \Sigma^* \to \Sigma^*$.
- Suppose that A is sparse: $\exists q \, \forall n \in \mathbb{N} : dens_A(n) \leq q(n)$.
- Since $p \in \mathbf{FP}$, $\exists \, r \in poly(n) : |p(x, y)| \leq r(|x| + |y|)$.
- Fix a $x \in A$, since p is 1-1 :

$$2^n \leq |\{p(x, y) : |y| \leq n\}| \leq dens_A(r(|x| + n)) \leq q(r(|x| + n))$$

Sparse Languages

Definition

A language L is *sparse* if there exists a polynomial q such that for every $n \in \mathbb{N} : dens_L(n) \leq q(n)$.

Theorem

If a language A is paddable, then it is not sparse.

Proof:

- Let $A \subseteq \Sigma^*$ with padding function $p : \Sigma^* \times \Sigma^* \to \Sigma^*$.
- Suppose that A is sparse: $\exists q \, \forall n \in \mathbb{N} : dens_A(n) \leq q(n)$.
- Since $p \in \mathbf{FP}$, $\exists \, r \in poly(n) : |p(x, y)| \leq r(|x| + |y|)$.
- Fix a $x \in A$, since p is 1-1 :

$$2^n \leq |\{p(x, y) : |y| \leq n\}| \leq dens_A(r(|x| + n)) \leq q(r(|x| + n))$$

- Thus, $2^n / q(r(|x| + n)) \leq 1$. Contradiction! $\square$

Sparse Languages

Theorem

If the Berman-Hartmanis conjecture is true, then all **NP***-complete and all co***NP***-complete languages are not sparse.*

Sparse Languages

Theorem

If the Berman-Hartmanis conjecture is true, then all **NP***-complete and all co***NP***-complete languages are not sparse.*

Proof:

- Berman-Hartmanis conjecture is true$\Rightarrow$ every **NP**-complete language A is polynomial-time isomorphic to SAT.
- Let f be this isomorphism, and pad_{SAT} a padding function for SAT.
- Define $p_A(x,y) := f^{-1}(pad_{\mathrm{SAT}}(f(x),y))$

Sparse Languages

Theorem

If the Berman-Hartmanis conjecture is true, then all **NP***-complete and all co***NP***-complete languages are not sparse.*

Proof:

- Berman-Hartmanis conjecture is true$\Rightarrow$ every **NP**-complete language A is polynomial-time isomorphic to SAT.
- Let f be this isomorphism, and pad_{SAT} a padding function for SAT.
- Define $p_A(x, y) := f^{-1}(pad_{\text{SAT}}(f(x), y))$
- Then $x \in A \Leftrightarrow f(x) \in \text{SAT} \Leftrightarrow pad_{\text{SAT}}(f(x), y) \in \text{SAT} \Leftrightarrow f^{-1}(pad_{\text{SAT}}(f(x), y)) \in A$.
- pad_{SAT} and f are polynomial time *computable* and *invertible*.

Sparse Languages

Proof (*cont'd*):

- So, p_A is a padding function for A, hence A is paddable.
- By the previous theorem, A is *not sparse*.

Sparse Languages

Proof (*cont'd*):

- So, p_A is a padding function for A, hence A is paddable.
- By the previous theorem, A is *not sparse*.
- Also, the complements of paddable languages are paddable (*why?*), so *co***NP**-complete languages are also not sparse. $\square$

Sparse Languages

Proof (*cont'd*):

- So, p_A is a padding function for A, hence A is paddable.
- By the previous theorem, A is *not sparse*.
- Also, the complements of paddable languages are paddable (*why?*), so *co*NP-complete languages are also not sparse. $\square$

Theorem

Suppose that a unary language $U \subseteq \{0\}^*$ is ***NP****-complete. Then,* $\boldsymbol{P} = \boldsymbol{NP}$.

Sparse Languages

Proof (*cont'd*):

- So, p_A is a padding function for A, hence A is paddable.
- By the previous theorem, A is *not sparse*.
- Also, the complements of paddable languages are paddable (*why?*), so *co***NP**-complete languages are also not sparse. $\square$

Theorem

*Suppose that a unary language $U \subseteq \{0\}^*$ is **NP**-complete. Then, **P** = **NP**.*

Theorem (Mahaney)

For any sparse $S \neq \emptyset$, SAT $\leq_m^p S$ if and only if $\mathbf{P} = \mathbf{NP}$.

Summary

- Classes like **NP**, **PSPACE** or **FP** can be *effectively enumerated*.
- If $\mathbf{P} \neq \mathbf{NP}$, there exist problems in **NP** which are not **NP**-complete neither in **P**.
- We can obtain polynomial-time isomorphisms between languages, given they are interreducible and paddable.
- Berman-Hartmanis Conjecture postulates that all NP-complete languages are polynomial-time isomorphic to each other.
- We can use padding to *translate upwards* equalities between complexity classes.
- If $\mathbf{P} \neq \mathbf{NP}$, then a *sparse* set *cannot* be $\leq_m^p$-hard for **NP**.

Contents

Warmup: Polynomial Identity Testing

1. Two polynomials are equal if they have the same coefficients for corresponding powers of their variable.
2. A polynomial is **identically zero** if all its coefficients are equal to the additive identity element.
3. How we can test if a polynomial is identically zero?

Warmup: Polynomial Identity Testing

1. Two polynomials are equal if they have the same coefficients for corresponding powers of their variable.
2. A polynomial is **identically zero** if all its coefficients are equal to the additive identity element.
3. How we can test if a polynomial is identically zero?
4. We can choose uniformly at random $r_1, \dots, r_n$ from a set $S \subseteq \mathbb{F}$.
5. We are wrong with a probability at most:

Theorem (Schwartz-Zippel Lemma)

Let $Q(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ be a multivariate polynomial of total degree d. Fix any finite set $S \subseteq \mathbb{F}$, and let $r_1, \dots, r_n$ be chosen indepedently and uniformly at random from S. Then:

$$\mathbf{Pr}[Q(r_1, \dots, r_n) = 0 | Q(x_1, \dots, x_n) \neq 0] \leq \frac{d}{|S|}$$

Warmup: Polynomial Identity Testing

Proof (*By Induction on n*):

- Base: $\mathbf{Pr}[Q(r) = 0 | Q(x) \neq 0] \leq d/|S|$
- Step:

$$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_n)$$

where $k \leq d$ is the *largest* exponent of x_1 in Q.
$deg(Q_k) \leq d - k \Rightarrow \mathbf{Pr}[Q_k(r_2, \ldots, r_n) = 0] \leq (d-k)/|S|$
Suppose that $Q_k(r_2, \ldots, r_n) \neq 0$. Then:

$$q(x_1) = Q(x_1, r_2, \ldots, r_n) = \sum_{i=0}^{k} x_1^i Q_i(r_2, \ldots, r_n)$$

$deg(q(x_1)) = k$, and $q(x_1) \neq 0$!

Warmup: Polynomial Identity Testing

Proof (*cont'd*):
The base case now implies that:

$$\mathbf{Pr}[q(r_1) = Q(r_1, \ldots, r_n) = 0] \leq k/|S|$$

Thus, we have shown the following two equalities:

$$\mathbf{Pr}[Q_k(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}$$

$$\mathbf{Pr}[Q_k(r_1, r_2, \ldots, r_n) = 0 | Q_k(r_2, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}$$

Using the following identity: $\mathbf{Pr}[\mathcal{E}_1] \leq \mathbf{Pr}[\mathcal{E}_1 | \overline{\mathcal{E}}_2] + \mathbf{Pr}[\mathcal{E}_2]$ we obtain that the requested probability is no more than the sum of the above, which proves our theorem! $\square$

Probabilistic Turing Machines

- A Probabilistic Turing Machine is a TM as we know it, but with access to a “random source”, that is an extra (read-only) tape containing *random-bits*!

Probabilistic Turing Machines

- A Probabilistic Turing Machine is a TM as we know it, but with access to a “random source”, that is an extra (read-only) tape containing *random-bits*!
- Randomization on:
  - **Output** (one or two-sided)
  - **Running Time**

Probabilistic Turing Machines

- A Probabilistic Turing Machine is a TM as we know it, but with access to a "random source", that is an extra (read-only) tape containing *random-bits*!
- Randomization on:
  - **Output** (one or two-sided)
  - **Running Time**

Definition (Probabilistic Turing Machines)

A Probabilistic Turing Machine is a TM with two transition functions δ_0, δ_1. On input x, we choose in each step with probability $1/2$ to apply the transition function δ_0 or δ_1, indepedently of all previous choices.

- We denote by $M(x)$ the *random variable* corresponding to the output of M at the end of the process.
- For a function $T : \mathbb{N} \to \mathbb{N}$, we say that M runs in $T(|x|)$-time if it halts on x within $T(|x|)$ steps (*regardless of the random choices it makes*).

BPP Class

Definition (BPP Class)

For $T : \mathbb{N} \to \mathbb{N}$, let $\mathbf{BPTIME}[T(n)]$ the class of languages L such that there exists a PTM which halts in $\mathcal{O}\left(T(|x|)\right)$ time on input x, and $\mathbf{Pr}[M(x) = L(x)] \geq 2/3$.
We define:

$$\mathbf{BPP} = \bigcup_{c \in \mathbb{N}} \mathbf{BPTIME}[n^c]$$

BPP Class

Definition (BPP Class)

For $T : \mathbb{N} \to \mathbb{N}$, let $\mathbf{BPTIME}[T(n)]$ the class of languages L such that there exists a PTM which halts in $\mathcal{O}\left(T(|x|)\right)$ time on input x, and $\mathbf{Pr}[M(x) = L(x)] \geq 2/3$.
We define:

$$\mathbf{BPP} = \bigcup_{c \in \mathbb{N}} \mathbf{BPTIME}[n^c]$$

- The class **BPP** represents our notion of **efficient** (*randomized*) computation!
- We can also define **BPP** using certificates:

BPP Class

Definition (Alternative Definition of BPP)

A language $L \in \mathbf{BPP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}_{r \in \{0,1\}^{p(n)}}[M(x,r) = L(x)] \geq \frac{2}{3}$$

BPP Class

Definition (Alternative Definition of BPP)

A language $L \in \mathbf{BPP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}_{r \in \{0,1\}^{p(n)}}[M(x,r) = L(x)] \geq \frac{2}{3}$$

- $\mathbf{P} \subseteq \mathbf{BPP}$

BPP Class

Definition (Alternative Definition of BPP)

A language $L \in \mathbf{BPP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}_{r \in \{0,1\}^{p(n)}}[M(x,r) = L(x)] \geq \frac{2}{3}$$

- $\mathbf{P} \subseteq \mathbf{BPP}$
- $\mathbf{BPP} \subseteq \mathbf{EXP}$ (*Trivial Derandomization*)

BPP Class

Definition (Alternative Definition of BPP)

A language $L \in \mathbf{BPP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}_{r \in \{0,1\}^{p(n)}}[M(x,r) = L(x)] \geq \frac{2}{3}$$

- $\mathbf{P} \subseteq \mathbf{BPP}$
- $\mathbf{BPP} \subseteq \mathbf{EXP}$ (*Trivial Derandomization*)
- The "**P** vs **BPP**" question.

Quantifier Characterizations

Definition (Majority Quantifier)

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a predicate, and ε a rational number, such that $\varepsilon \in \left(0, \frac{1}{2}\right)$. We denote by $(\exists^+ y, |y| = k)R(x,y)$ the following predicate:

> *"There exist at least $\left(\frac{1}{2} + \varepsilon\right) \cdot 2^k$ strings y of length m for which $R(x,y)$ holds."*

We call $\exists^+$ the *overwhelming majority* quantifier.

- $\exists_r^+$ means that the fraction r of the possible certificates of a certain length satisfy the predicate for the certain input.

Quantifier Characterizations

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x, y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x, y)$

Quantifier Characterizations

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x, y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x, y)$

- $\mathbf{P} = (\forall/\forall)$
- $\mathbf{NP} = (\exists/\forall)$
- $co\mathbf{NP} = (\forall/\exists)$
- $\mathbf{BPP} = (\exists^+/\exists^+) = co\mathbf{BPP}$

RP Class

- In the same way, we can define classes that contain problems with one-sided error:

Definition

The class $\mathbf{RTIME}[T(n)]$ contains every language L for which there exists a PTM M running in $\mathcal{O}(T(|x|))$ time such that:

- $x \in L \Rightarrow \mathbf{Pr}[M(x) = 1] \geq \frac{2}{3}$
- $x \notin L \Rightarrow \mathbf{Pr}[M(x) = 0] = 1$

We define

$$\mathbf{RP} = \bigcup_{c \in \mathbb{N}} \mathbf{RTIME}[n^c]$$

- Similarly we define the class *co*$\mathbf{RP}$.

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall)$

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$ (*every accepting path is a certificate!*)

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$ (*every accepting path is a certificate!*)
- $co\mathbf{RP} = (\forall/\exists^+) \subseteq (\forall/\exists) = co\mathbf{NP}$

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$ (*every accepting path is a certificate!*)
- $co\mathbf{RP} = (\forall/\exists^+) \subseteq (\forall/\exists) = co\mathbf{NP}$

Theorem (Decisive Characterization of BPP)

$$\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$$

- The above characterization is **decisive**, in the sense that if we replace $\exists^+$ with $\exists$, the two predicates are still complementary (i.e. $R_1 \Rightarrow \neg R_2$), so they still define a complexity class.
- In the above characterization of **BPP**, if we replace $\exists^+$ with $\exists$, we obtain very easily a well-known result:

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{RP} = (\exists^+/\forall) \subseteq (\exists/\forall) = \mathbf{NP}$ (*every accepting path is a certificate!*)
- $co\mathbf{RP} = (\forall/\exists^+) \subseteq (\forall/\exists) = co\mathbf{NP}$

Theorem (Decisive Characterization of BPP)

$$\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$$

- The above characterization is **decisive**, in the sense that if we replace $\exists^+$ with $\exists$, the two predicates are still complementary (i.e. $R_1 \Rightarrow \neg R_2$), so they still define a complexity class.
- In the above characterization of **BPP**, if we replace $\exists^+$ with $\exists$, we obtain very easily a well-known result:

Corollary (Sipser-Gács Theorem)

$$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

Error Reduction for BPP

- How important is $2/3$?

Error Reduction for BPP

- How important is $2/3$?

Theorem (Error Reduction for BPP)

Let $L \subseteq \{0,1\}^*$ be a language and suppose that there exists a poly-time PTM M such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$$

Then, for every constant $d > 0$, $\exists$ poly-time PTM M' such that for every $x \in \{0,1\}^*$:

$$\mathbf{Pr}[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

Proof: Th.7.10 (p.132) in [2]

- The machine M' does the following:
    - Run $M(x)$ for every input x for $k = 8|x|^{2c+d}$ times, and obtain outputs $y_1, y_2, \ldots, y_k \in \{0, 1\}$.
    - If the majority of these outputs is 1, return 1
    - Otherwise, return 0.

Proof: Th.7.10 (p.132) in [2]

- The machine M' does the following:
  - Run $M(x)$ for every input x for $k = 8|x|^{2c+d}$ times, and obtain outputs $y_1, y_2, \dots, y_k \in \{0, 1\}$.
  - If the majority of these outputs is 1, return 1
  - Otherwise, return 0.
- We define the r.v. X_i for every $i \in [k]$ to be 1 if $y_i = L(x)$ and 0 otherwise.
- $X_1, X_2, \dots, X_k$ are indepedent Boolean random variables, with:

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] \geq p = \frac{1}{2} + |x|^{-c}$$

Proof: Th.7.10 (p.132) in [2]

- The machine M' does the following:
  - Run $M(x)$ for every input x for $k = 8|x|^{2c+d}$ times, and obtain outputs $y_1, y_2, \dots, y_k \in \{0,1\}$.
  - If the majority of these outputs is 1, return 1
  - Otherwise, return 0.
- We define the r.v. X_i for every $i \in [k]$ to be 1 if $y_i = L(x)$ and 0 otherwise.
- $X_1, X_2, \dots, X_k$ are indepedent Boolean random variables, with:

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] \geq p = \frac{1}{2} + |x|^{-c}$$

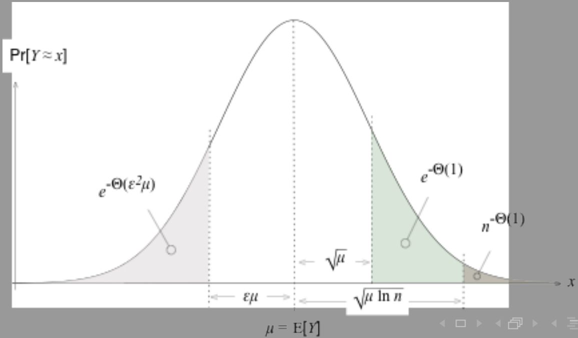
- Applying a Chernoff Bound we obtain:

$$\mathbf{Pr}\left[|\sum_{i=1}^{k} X_i - pk| > \delta pk \right] < e^{-\frac{\delta^2}{4}pk} = e^{-\frac{1}{4|x|^{2c}}\frac{1}{2}8|x|^{2c+d}} \leq 2^{-|x|^d}$$

□

Intermission: Chernoff Bounds

- *How many* samples do we need in order to estimate μ up to an *error* of $\pm\varepsilon$ with *probability* at least $1-\delta$?
- Chernoff Bound tells us that this number is $\mathcal{O}\left(\rho/\varepsilon^2\right)$, where $\rho = \log(1/\delta)$.
- The probability that k is $\rho\sqrt{n}$ far from μn decays **exponentially** with ρ.



Intermission: Chernoff Bounds

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}$$

$$\mathbf{Pr}\left[\sum_{i=1}^{n} X_i \leq (1-\delta)\mu\right] \leq \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu}$$

Other useful form is:

$$\mathbf{Pr}\left[|\sum_{i=1}^{n} X_i - \mu| \geq c\mu\right] \leq 2e^{-\min\{c^2/4, c/2\}\cdot\mu}$$

- This probability is bounded by $2^{-\Omega(\mu)}$.

Error Reduction for BPP

- From the above we can obtain the following interesting corollary:

Corollary

For $c > 0$, let $\mathbf{BPP}_{1/2+n^{-c}}$ denote the class of languages L for which there is a polynomial-time PTM M satisfying $\mathbf{Pr}[M(x) = L(x)] \geq 1/2 + |x|^{-c}$ for every $x \in \{0,1\}^*$.Then:

$$\mathbf{BPP}_{1/2+n^{-c}} = \mathbf{BPP}$$

- Obviously, $\exists^+ = \exists^+_{1/2+\varepsilon} = \exists^+_{2/3} = \exists^+_{3/4} = \exists^+_{0.99} = \exists^+_{1-2^{-p(|x|)}}$

ZPP Class

- And now something completely different:
- What if the random variable was the *running time* and not the output?

ZPP Class

- And now something completely different:
- What if the random variable was the *running time* and not the output?
- We say that M has expected running time $T(n)$ if the expectation $\mathbf{E}[T_{M(x)}]$ is at most $T(|x|)$ for every $x \in \{0,1\}^*$.
  ($T_{M(x)}$ is the running time of M on input x, and it is a **random variable**!)

Definition

The class $\mathbf{ZTIME}[T(n)]$ contains all languages L for which there exists a machine M that runs in an expected time $\mathcal{O}\left(T(|x|)\right)$ such that for every input $x \in \{0,1\}^*$, whenever M halts on x, the output $M(x)$ it produces is exactly $L(x)$. We define:

$$\mathbf{ZPP} = \bigcup_{c \in \mathbb{N}} \mathbf{ZTIME}[n^c]$$

ZPP Class

- The output of a **ZPP** machine is **always** correct!
- The problem is that we aren't sure about the running time.
- We can easily see that $\mathbf{ZPP} = \mathbf{RP} \cap co\mathbf{RP}$.

- The next Hasse diagram summarizes the previous inclusions: (Recall that $\Delta\Sigma_2^p = \Sigma_2^p \cap \Pi_2^p = \mathbf{NP}^{\mathbf{NP}} \cap co\mathbf{NP}^{\mathbf{NP}}$)

Zero-Error

$$\begin{tikzcd}
 & \mathbf{PSPACE} & \\
 & \Delta\Sigma_2^p \arrow[u] & \\
co\mathbf{NP} \arrow[ur] & & \mathbf{NP} \arrow[ul] \\
 & \mathbf{BPP} \arrow[uu] & \\
co\mathbf{RP} \arrow[uu] \arrow[ur] & & \mathbf{RP} \arrow[uu] \arrow[ul] \\
 & \mathbf{ZPP} \arrow[ul] \arrow[ur] & \\
 & \mathbf{P} \arrow[u] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & \mathbf{PSPACE} & & \\
 & & (\exists\forall/\forall\exists) \cap (\forall\exists/\exists\forall) \arrow[u] & & \\
(\forall/\exists) \arrow[urr] & & & & (\exists/\forall) \arrow[ull] \\
 & & {\color{blue}(\exists^+/\exists^+)} \arrow[uu] & & \\
{\color{blue}(\forall/\exists^+)} \arrow[uu] \arrow[urr] & & & & {\color{blue}(\exists^+/\forall)} \arrow[uu] \arrow[ull] \\
 & & {\color{blue}(\forall/\forall)} \arrow[ull] \arrow[urr] & &
\end{tikzcd}$$

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
  $x \in L \Leftrightarrow$ #accepting paths $\neq 0$

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
  $x \in L \Leftrightarrow$ #accepting paths $\neq 0$
- We can get an effective enumeration of all NPTMs, each deciding an **NP** language.

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
  $x \in L \Leftrightarrow$ #accepting paths $\neq 0$
- We can get an effective enumeration of all NPTMs, each deciding an **NP** language.
- But not every NPTM decides a language in **RP**:
  e.g., the NPTM that has *exactly one* accepting path.

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
  $x \in L \Leftrightarrow$ #accepting paths $\neq 0$
- We can get an effective enumeration of all NPTMs, each deciding an **NP** language.
- But <u>not</u> every NPTM decides a language in **RP**:
  e.g., the NPTM that has *exactly one* accepting path.
- In this case, there is no way to tell whether the machine will always halt with the certified output. We call these classes **semantic**.

Semantic vs. Syntactic Classes

- Every NPTM defines some language in **NP**:
  $x \in L \Leftrightarrow \#\text{accepting paths} \neq 0$
- We can get an effective enumeration of all NPTMs, each deciding an **NP** language.
- But not every NPTM decides a language in **RP**:
  e.g., the NPTM that has *exactly one* accepting path.
- In this case, there is no way to tell whether the machine will always halt with the certified output. We call these classes **semantic**.
- So we have:
  - **Syntactic Classes** (like **P**, **NP**)
  - **Semantic Classes** (like **RP**, **BPP**, **NP** $\cap$ *co***NP**, **TFNP**)

Complete Problems for BPP?

- Any syntactic class has a “free” complete problem:

$$\{\langle M, x\rangle : M \in \mathcal{M} \;\&\; M(x) = \text{``}yes''\}$$

where $\mathcal{M}$ is the class of TMs of the variant that defines the class

- In semantic classes, this complete language is usually *undecidable* (Rice’s Theorem).
- The defining property of **BPTIME** machines is **semantic**!

Complete Problems for BPP?

- Any syntactic class has a "free" complete problem:

$$\{\langle M, x\rangle : M \in \mathcal{M} \ \& \ M(x) = \text{``}yes''\}$$

  where $\mathcal{M}$ is the class of TMs of the variant that defines the class
- In semantic classes, this complete language is usually *undecidable* (Rice's Theorem).
- The defining property of **BPTIME** machines is **semantic**!
- If finally $\mathbf{P} = \mathbf{BPP}$, then **BPP** will have complete problems!!
- For the same reason, in semantic classes we cannot prove Hierarchy Theorems using Diagonalization.

The Class PP

Definition

A language $L \in \mathbf{PP}$ if there exists an NPTM M, such that for every $x \in \{0,1\}^*$: $x \in L$ if and only if *more than half* of the computations of M on input x accept.

- Or, equivalently:

Definition

A language $L \in \mathbf{PP}$ if there exists a poly-time TM M and a polynomial $p \in poly(n)$, such that for every $x \in \{0,1\}^*$:

$$x \in L \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\right\}\right| \geq \frac{1}{2} \cdot 2^{p(|x|)}$$

The Class PP

- The defining property of **PP** is **syntactic**, any NPTM can define a language in **PP**.
- Due to the lack of a gap between the two cases, we cannot amplify the probability with polynomially many repetitions, as in the case of **BPP**.
- **PP** is closed under complement.
- A breakthrough result of R. Beigel, N. Reingold and D. Spielman is that **PP** is closed under *intersection*!

The Class PP

- The defining property of **PP** is **syntactic**, any NPTM can define a language in **PP**.
- Due to the lack of a gap between the two cases, we cannot amplify the probability with polynomially many repetitions, as in the case of **BPP**.
- **PP** is closed under complement.
- A breakthrough result of R. Beigel, N. Reingold and D. Spielman is that **PP** is closed under *intersection*!
- The syntactic definition of **PP** gives the possibility for *complete problems*:
- Consider the problem MAJSAT:
  Given a Boolean Expression, is it true that the majority of the 2^n truth assignments to its variables (that is, at least $2^{n-1}+1$ of them) satisfy it?

The Class PP

Theorem

MAJSAT is **PP***-complete!*

- MAJSAT is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

The Class PP

Theorem

MAJSAT is **PP***-complete!*

- MAJSAT is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

Theorem

$$\mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$$

The Class PP

Theorem

MAJSAT is **PP***-complete!*

- MAJSAT is not likely in **NP**, since the (*obvious*) certificate is not very succinct!

Theorem

$$\mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$$

Proof: Th.11.3 (p.257) in [1]

It is easy to see that **PP** $\subseteq$ **PSPACE**:
We can simulate any **PP** machine by enumerating all strings y of length $p(n)$ and verify whether **PP** machine accepts. The **PSPACE** machine accepts if and only if there are more than $2^{p(n)-1}$ such y's (by using a counter).

The Class PP

Proof (*cont'd*):
Now, for **NP** $\subseteq$ **PP**, let $A \in$ **NP**. That is, $\exists p \in poly(n)$ and a poly-time and balanced predicate R such that:

$$x \in A \;\Leftrightarrow\; (\exists y, |y| = p(|x|)) : R(x, y)$$

The Class PP

Proof (*cont'd*):
Now, for **NP** $\subseteq$ **PP**, let $A \in$ **NP**. That is, $\exists p \in poly(n)$ and a poly-time and balanced predicate R such that:

$$x \in A \;\Leftrightarrow\; (\exists y, |y| = p(|x|)) : R(x, y)$$

Consider the following TM:

M accepts input (x, by), with $|b| = 1$ and $|y| = p(|x|)$, if and only if $R(x, y) = 1$ or $b = 1$.

The Class PP

Proof (*cont'd*):
Now, for **NP** $\subseteq$ **PP**, let $A \in$ **NP**. That is, $\exists p \in poly(n)$ and a poly-time and balanced predicate R such that:

$$x \in A \;\Leftrightarrow\; (\exists y, |y| = p(|x|)) : R(x, y)$$

Consider the following TM:

M accepts input (x, by), with $|b| = 1$ and $|y| = p(|x|)$, if and only if $R(x, y) = 1$ or $b = 1$.

- If $x \in A$, then $\exists$ at least one y s.t. $R(x, y)$.
  Thus, $\mathbf{Pr}[M(x)\text{ accepts}] \geq 1/2 + 2^{-(p(n)+1)}$.
- If $x \notin A$, then $\mathbf{Pr}[M(x)\text{ accepts}] = 1/2$. $\square$

Other Results

Theorem

If $\mathbf{NP} \subseteq \mathbf{BPP}$, *then* $\mathbf{NP} = \mathbf{RP}$.

Other Results

Theorem

If $\mathbf{NP} \subseteq \mathbf{BPP}$, *then* $\mathbf{NP} = \mathbf{RP}$.

Proof:

- **RP** is closed under $\leq_m^p$-reducibility.
- It suffices to show that if SAT $\in$ **BPP**, then SAT $\in$ **RP**.
- Recall that SAT has the **self-reducibility** property: $\phi(x_1, \ldots, x_n)$: $\phi \in \text{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \text{SAT} \ \vee \ \phi|_{x_1=1} \in \text{SAT})$.
- SAT $\in$ **BPP**: $\exists$ PTM M computing SAT with error probability bounded by $2^{-|\phi|}$.
- We can use the *self-reducibility* of SAT to produce a truth assignment for ϕ as follows:

Other Results

Proof (*cont'd*):

```
Input: A Boolean formula $\phi$ with $n$ variables
If $M(\phi) = 0$ then reject $\phi$;
For $i = 1$ to $n$
→ If $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=0}) = 1$ then let $\alpha_i = 0$
→ ElseIf $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=1}) = 1$ then let $\alpha_i = 1$
→ Else reject $\phi$ and halt;
If $\phi|_{x_1=\alpha_1,\ldots,x_n=\alpha_n} = 1$ then accept $F$
Else reject $F$
```

Other Results

Proof (*cont'd*):

```
Input: A Boolean formula $\phi$ with $n$ variables
If $M(\phi) = 0$ then reject $\phi$;
For $i = 1$ to $n$
→ If $M(\phi|_{x_1=\alpha_1,\dots,x_{i-1}=\alpha_{i-1},x_i=0}) = 1$ then let $\alpha_i = 0$
→ ElseIf $M(\phi|_{x_1=\alpha_1,\dots,x_{i-1}=\alpha_{i-1},x_i=1}) = 1$ then let $\alpha_i = 1$
→ Else reject $\phi$ and halt;
If $\phi|_{x_1=\alpha_1,\dots,x_n=\alpha_n} = 1$ then accept $F$
Else reject $F$
```

- Note that M_1 accepts ϕ *only if* a t.a. $t(x_i) = \alpha_i$ is found.
- Therefore, M_1 never makes mistakes if $\phi \notin \mathrm{SAT}$.
- If $\phi \in \mathrm{SAT}$, then M rejects ϕ on each iteration of the loop w.p. $2^{-|\phi|}$.
- So, $\mathbf{Pr}[M_1 \text{ accepting } x] = (1 - 2^{-|\phi|})^n$, which is greater than $1/2$ if $|\phi| \geq n > 1$. $\square$

Relativized Results

Theorem

Relative to a random oracle A, $\mathbf{P}^A = \mathbf{BPP}^A$. That is,

$$\mathbf{Pr}_{A\in\{0,1\}^*}[\mathbf{P}^A = \mathbf{BPP}^A] = 1$$

Also,

- $\mathbf{BPP}^A \subsetneq \mathbf{NP}^A$, relative to a *random* oracle A.
- There exists an A such that: $\mathbf{P}^A \neq \mathbf{RP}^A$.
- There exists an A such that: $\mathbf{RP}^A \neq co\mathbf{RP}^A$
- There exists an A such that: $\mathbf{RP}^A \neq \mathbf{NP}^A$.

Relativized Results

Theorem

Relative to a random oracle A, $\mathbf{P}^A = \mathbf{BPP}^A$. *That is,*

$$\mathbf{Pr}_{A\in\{0,1\}^*}[\mathbf{P}^A = \mathbf{BPP}^A] = 1$$

Also,

- $\mathbf{BPP}^A \subsetneq \mathbf{NP}^A$, relative to a *random* oracle A.
- There exists an A such that: $\mathbf{P}^A \neq \mathbf{RP}^A$.
- There exists an A such that: $\mathbf{RP}^A \neq co\mathbf{RP}^A$
- There exists an A such that: $\mathbf{RP}^A \neq \mathbf{NP}^A$.

Corollary

There exists an A such that:

$$\mathbf{P}^A \neq \mathbf{RP}^A \neq \mathbf{NP}^A \not\subseteq \mathbf{BPP}^A$$

Summary

- Randomized Computation uses random bits, and either the *output* is a random variable (**BPP** for two-sided and **RP** for one-sided error) or the *running time* (**ZPP**).
- The error for **BPP** and **RP** can be *reduced* to be exponentially close to 0, by polynomially many repetitions.
- **BPP** is in the second level of **PH**.
- $\mathbf{ZPP} = \mathbf{RP} \cap co\mathbf{RP}$.
- *Semantic* classes like **BPP**, **RP**, **ZPP** don't seem to have complete problems.

Contents

Boolean Circuits

- A Boolean Circuit is a natural model of *nonuniform* computation, a generalization of hardware computational methods.

- A **non-uniform** computational model allows us to use a different "algorithm" to be used for every input size, in contrast to the standard (or *uniform*) Turing Machine model, where the same T.M. is used on (infinitely many) input sizes.

- Each circuit can be used for a **fixed** input size, which limits or model.

Definition (Boolean circuits)

For every $n \in \mathbb{N}$ an n-input, single output Boolean Circuit C is a directed acyclic graph with n sources and *one* sink.

- All nonsource vertices are called *gates* and are labeled with one of $\wedge$ (and), $\vee$ (or) or $\neg$ (not).
- The vertices labeled with $\wedge$ and $\vee$ have *fan-in* (i.e. number or incoming edges) 2.
- The vertices labeled with $\neg$ have *fan-in* 1.
- The *size* of C, denoted by $|C|$, is the number of vertices in it.
- For every vertex v of C, we assign a value as follows: for some input $x \in \{0,1\}^n$, if v is the i-th input vertex then $val(v) = x_i$, and otherwise $val(v)$ is defined recursively by applying v's logical operation on the values of the vertices connected to v.
- The *output* $C(x)$ is the value of the output vertex.
- The *depth* of C is the length of the longest directed path from an input node to the output node.

- To overcome the fixed input length size, we need to allow families (or sequences) of circuits to be used:

Definition

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A *$T(n)$-size circuit family* is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.

- To overcome the fixed input length size, we need to allow families (or sequences) of circuits to be used:

Definition

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$*-size circuit family* is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.

- These infinite families of circuits are defined arbitrarily: There is **no** pre-defined connection between the circuits, and also we haven't any "guarantee" that we can construct them efficiently.

- To overcome the fixed input length size, we need to allow families (or sequences) of circuits to be used:

Definition

Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be a function. A $T(n)$*-size circuit family* is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.

- These infinite families of circuits are defined arbitrarily: There is **no** pre-defined connection between the circuits, and also we haven't any "guarantee" that we can construct them efficiently.
- Like each new computational model, we can define a complexity class on it by imposing some restriction on a *complexity measure*:

Definition

We say that a language L is in $\mathbf{SIZE}[T(n)]$ if there is a $T(n)$-*size circuit family* $\{C_n\}_{n\in\mathbb{N}}$, such that $\forall x \in \{0,1\}^n$:

$$x \in L \Leftrightarrow C_n(x) = 1$$

Definition

We say that a language L is in $\mathbf{SIZE}[T(n)]$ if there is a $T(n)$*-size circuit family* $\{C_n\}_{n\in\mathbb{N}}$, such that $\forall x \in \{0,1\}^n$:

$$x \in L \Leftrightarrow C_n(x) = 1$$

Definition

$\mathbf{P}_{/\mathbf{poly}}$ is the class of languages that are decidable by polynomial size circuits families:

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c\in\mathbb{N}} \mathbf{SIZE}[n^c]$$

Definition

We say that a language L is in $\mathbf{SIZE}[T(n)]$ if there is a $T(n)$-*size circuit family* $\{C_n\}_{n\in\mathbb{N}}$, such that $\forall x \in \{0,1\}^n$:

$$x \in L \Leftrightarrow C_n(x) = 1$$

Definition

$\mathbf{P}_{/\mathbf{poly}}$ is the class of languages that are decidable by polynomial size circuits families:

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c\in\mathbb{N}} \mathbf{SIZE}[n^c]$$

Theorem (Nonuniform Hierarchy Theorem)

For every functions $T, T' : \mathbb{N} \to \mathbb{N}$ with $\frac{2^n}{n} > T'(n) > 10T(n) > n$,

$$\mathbf{SIZE}[T(n)] \subsetneq \mathbf{SIZE}[T'(n)]$$

Turing Machines that take advice

Definition

Let $T, a : \mathbb{N} \rightarrow \mathbb{N}$. The class of languages decidable by $T(n)$-time Turing Machines with $a(n)$ bits of advice, denoted

$$\mathbf{DTIME}[T(n)/a(n)]$$

contains every language L such that there exists a sequence $\{d_n\}_{n\in\mathbb{N}}$ of strings, with $d_n \in \{0,1\}^{a(n)}$ and a Turing Machine M satisfying:

$$x \in L \Leftrightarrow M(x, d_n) = 1$$

for every $x \in \{0,1\}^n$, where on input (x, d_n) the machine M runs for at most $\mathcal{O}(T(n))$ steps.

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,k\in\mathbb{N}} \mathbf{DTIME}[n^c/n^k]$$

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,k\in\mathbb{N}} \mathbf{DTIME}[n^c/n^k]$$

Proof: ($\subseteq$) Let $L \in \mathbf{P}_{/\mathbf{poly}}$. Then, $\exists\{C_n\}_{n\in\mathbb{N}} : C_{|x|} = L(x)$.
We can use C_n 's encoding as an advice string for each n.

Turing Machines that take advice

Theorem (Alternative Definition of $\mathbf{P}_{/\mathbf{poly}}$)

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,k\in\mathbb{N}} \mathbf{DTIME}[n^c/n^k]$$

Proof: ($\subseteq$) Let $L \in \mathbf{P}_{/\mathbf{poly}}$. Then, $\exists\{C_n\}_{n\in\mathbb{N}} : C_{|x|} = L(x)$.
We can use C_n 's encoding as an advice string for each n.
($\supseteq$) Let $L \in \mathbf{DTIME}[n^c/n^k]$. Then, since CVP is **P**-complete, we construct for every n a circuit D_n such that, for $x \in \{0,1\}^n, d_n \in \{0,1\}^{a(n)}$:

$$D_n(x, d_n) = M(x, d_n)$$

Then, let $C_n(x) = D_n(x, d_n)$ (**We hard-wire the advice string!**)
Since $a(n) = n^k$, the circuits have polynomial size. $\square$

Theorem

$$\mathbf{P} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

- For the subset inclusion, recall that CVP is **P**-complete.

Theorem

$$\mathbf{P} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

- For the subset inclusion, recall that CVP is **P**-complete.
- **But why proper inclusion?**

Theorem

$$\mathbf{P} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

- For the subset inclusion, recall that CVP is **P**-complete.
- **But why proper inclusion?**
- Consider the following language: $\mathrm{U} = \{1^n | n \in \mathbb{N}\}$.
- $\mathrm{U} \in \mathbf{P}_{/\mathbf{poly}}$.

Theorem

$$\mathbf{P} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

- For the subset inclusion, recall that CVP is **P**-complete.
- **But why proper inclusion?**
- Consider the following language: $\mathrm{U} = \{1^n | n \in \mathbb{N}\}$.
- $\mathrm{U} \in \mathbf{P}_{/\mathbf{poly}}$.
- Now consider this:

$$\mathrm{U_H} = \{1^n | \textit{n's binary expression encodes a pair } \llcorner M, x \lrcorner \textit{ s.t. } M(x) \downarrow\}$$

- It is easy to see that $\mathrm{U_H} \in \mathbf{P}_{/\mathbf{poly}}$, but....

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{PH} = \Sigma_2^p$.

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{PH} = \Sigma_2^p$.

Proof Sketch:

- It suffices to show that $\Pi_2^p \subseteq \Sigma_2^p$.
  (Recall that $\Sigma_2^p = \Pi_2^p \Rightarrow \mathbf{PH} = \Sigma_2^p$)
- Let $L \in \Pi_2^p$. Then, $x \in L \Rightarrow \forall y \exists z\, R(x, y, z)$

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$*, then* $\mathbf{PH} = \Sigma_2^p$.

Proof Sketch:

- It suffices to show that $\Pi_2^p \subseteq \Sigma_2^p$.
  (Recall that $\Sigma_2^p = \Pi_2^p \Rightarrow \mathbf{PH} = \Sigma_2^p$)
- Let $L \in \Pi_2^p$. Then, $x \in L \Rightarrow \forall y \underbrace{\exists z\, R(x, y, z)}_{\text{SAT Question}}$

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{PH} = \Sigma_2^p$.

Proof Sketch:

- It suffices to show that $\Pi_2^p \subseteq \Sigma_2^p$.
  (Recall that $\Sigma_2^p = \Pi_2^p \Rightarrow \mathbf{PH} = \Sigma_2^p$)
- Let $L \in \Pi_2^p$. Then, $x \in L \Rightarrow \forall y \underbrace{\exists z\, R(x, y, z)}_{\text{SAT Question}}$
- So, we can get a function $\phi(x, y) \in \mathbf{FP}$ s.t. :

$$x \in L \Leftrightarrow \forall y[\phi(x, y) \in \text{SAT}]$$

- Since SAT $\in \mathbf{P}_{/\mathbf{poly}}$, $\exists \{C_n\}_{n \in \mathbb{N}}$ s.t. $C_{|\phi|}(\phi(x, y)) = 1$ iff ϕ satisfiable.
- The idea is to nondeterministically *guess* such a circuit:

- If $x \in L$: $\boxed{\text{Since } L \in \Pi_2^p, x \in L \Rightarrow \forall y[\phi(x, y) \in \text{SAT}]}$
  We will guess a correct C, and $\forall y\ \phi(x, y)$ will be satisfiable, so C will accept all y's:

$$x \in L \Rightarrow \exists C\, \forall y\, [C(\phi(x, y)) = 1]$$

- If $x \in L$: $\boxed{\text{Since } L \in \Pi_2^p, x \in L \Rightarrow \forall y[\phi(x,y) \in \text{SAT}]}$
  We will guess a correct C, and $\forall y\ \phi(x,y)$ will be satisfiable, so C will accept all y's:
  $$x \in L \Rightarrow \exists C\, \forall y\, [C(\phi(x,y)) = 1]$$
- If $x \notin L$: $\boxed{\text{Since } L \in \Pi_2^p, x \notin L \Rightarrow \exists y[\phi(x,y) \notin \text{SAT}]}$
  Then, there will be a y_0 for which $\phi(x,y_0)$ is *not* satisfiable. So, for all guesses of C, $\phi(x,y_0)$ will always be rejected:
  $$x \notin L \Rightarrow \forall C\, \exists y\, [C(\phi(x,y)) = 0]$$
- That is a Σ_2^p question, so $L \in \Sigma_2^p \Rightarrow \Pi_2^p \subseteq \Sigma_2^p$. $\square$

- If $x \in L$: $\boxed{\text{Since } L \in \Pi_2^p,\ x \in L \Rightarrow \forall y[\phi(x,y) \in \text{SAT}]}$
  We will guess a correct C, and $\forall y\ \phi(x,y)$ will be satisfiable, so C will accept all y's:
  $$x \in L \Rightarrow \exists C\, \forall y\ [C(\phi(x,y)) = 1]$$
- If $x \notin L$: $\boxed{\text{Since } L \in \Pi_2^p,\ x \notin L \Rightarrow \exists y[\phi(x,y) \notin \text{SAT}]}$
  Then, there will be a y_0 for which $\phi(x,y_0)$ is *not* satisfiable. So, for all guesses of C, $\phi(x,y_0)$ will always be rejected:
  $$x \notin L \Rightarrow \forall C\, \exists y\ [C(\phi(x,y)) = 0]$$
- That is a Σ_2^p question, so $L \in \Sigma_2^p \Rightarrow \Pi_2^p \subseteq \Sigma_2^p$. $\square$

Theorem (Meyer's Theorem)

If $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{EXP} = \Sigma_2^p$.

Theorem

$$\mathbf{BPP} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

Theorem

$$\mathbf{BPP} \subsetneq \mathbf{P}_{/\mathbf{poly}}$$

Proof: Recall that if $L \in \mathbf{BPP}$, then $\exists$ PTM M such that:

$$\mathbf{Pr}_{r \in \{0,1\}^{poly(n)}} [M(x,r) \neq L(x)] < 2^{-n}$$

Then, taking the union bound:

$$\mathbf{Pr}[\exists x \in \{0,1\}^n : M(x,r) \neq L(x)] = \mathbf{Pr}\left[\bigcup_{x \in \{0,1\}^n} M(x,r) \neq L(x)\right] \leq$$

$$\leq \sum_{x \in \{0,1\}^n} \mathbf{Pr}[M(x,r) \neq L(x)] < 2^{-n} + \cdots + 2^{-n} = 1$$

So, $\exists r_n \in \{0,1\}^{poly(n)}$, s.t. $\forall x \{0,1\}^n$: $M(x,r_n) = L(x)$.
Using $\{r_n\}_{n \in \mathbb{N}}$ as advice string, we have the non-uniform machine. $\square$

Intermission: What kind of proof was that?

- How did we prove the previous theorem?

Intermission: What kind of proof was that?

- How did we prove the previous theorem?
- We constructed implicitily a probability space around an object we wish to prove its existence.

Intermission: What kind of proof was that?

- How did we prove the previous theorem?
- We constructed implicitily a probability space around an object we wish to prove its existence.
- If we randomly choose an existing object, the probability that the result is of the prescribed kind is > 0.
- That technique is called **The Probabilistic Method**.
- In the same way, showing that the probability is < 1 proves the existence of an object that *does not* satisfy the prescribed properties.

See: Noga Alon, Joel H. Spencer, **The Probabilistic Method**, 4th Edition, Wiley Publishing, 2016

Theorem

The following are equivalent:

1. $A \in \mathbf{P}_{/\mathbf{poly}}$.
2. *There exists a sparse set S such that $A \in \mathbf{P}^S$ (or $A \leq_T^p S$).*

Theorem

The following are equivalent:

1. $A \in \mathbf{P}_{/\mathbf{poly}}$.
2. *There exists a sparse set S such that $A \in \mathbf{P}^S$ (or $A \leq_T^p S$).*

Proof:

$(2) \Rightarrow (1)$

- Let $A \in \mathbf{P}^S$, and M the machine that decides it.
- On inputs of lenght n, there are at most *polynomially* many strings in S that can be queried by M in polynomial time.
- We *hard-wire* these strings in M, and transform it into a circuit.

Theorem

The following are equivalent:

1. $A \in \mathbf{P}_{/\mathbf{poly}}$.
2. *There exists a sparse set S such that $A \in \mathbf{P}^S$ (or $A \leq_T^p S$).*

Proof (*cont'd*):

$(1) \Rightarrow (2)$

- If $A \in \mathbf{P}_{/\mathbf{poly}}$, by using an advice function d, we can encode $d(n)$ as a *sparse* oracle:

$$S = \{\langle 1^n, p_n \rangle \mid p_n \text{ is a prefix of } d(n), n \geq 0\}$$

- We can retrieve the advice string by iteratively querying the oracle:
  - At first query $\langle 1^n, 0 \rangle, \langle 1^n, 1 \rangle$.
  - Then, for a prefix p we query $\langle 1^n, p0 \rangle, \langle 1^n, p1 \rangle$ etc... $\square$

Algorithms for Circuits

Definition (Circuit Complexity or Worst-Case Hardness)

For a finite Boolean Function $f : \{0,1\}^n \to \{0,1\}$, we define the (circuit) *complexity* of f, denoted $CC(f)$, as the size of the smallest Boolean Circuit computing f (that is, $C(x) = f(x), \forall x \in \{0,1\}^n$).

Algorithms for Circuits

Definition (Circuit Complexity or Worst-Case Hardness)

For a finite Boolean Function $f : \{0,1\}^n \to \{0,1\}$, we define the (circuit) *complexity* of f, denoted $CC(f)$, as the size of the smallest Boolean Circuit computing f (that is, $C(x) = f(x), \forall x \in \{0,1\}^n$).

Definition (MCSP)

Given the truth table of a Boolean function f and an integer S, does $CC(f) \leq S$?

Algorithms for Circuits

Definition (Circuit Complexity or Worst-Case Hardness)

For a finite Boolean Function $f : \{0,1\}^n \to \{0,1\}$, we define the (circuit) *complexity* of f, denoted $CC(f)$, as the size of the smallest Boolean Circuit computing f (that is, $C(x) = f(x), \forall x \in \{0,1\}^n$).

Definition (MCSP)

Given the truth table of a Boolean function f and an integer S, does $CC(f) \leq S$?

Definition (CAPP)

Given circuit C and a constant $\varepsilon > 0$, output u such that: $|\mathbf{Pr}_x [C(x) = 1] - u| < \varepsilon$.

Algorithms for Circuits

- MCSP $\in$ **NP**.
- But, MCSP *doesn't* seem to be **NP**-complete.
  (*Murray, Williams, 2017*)

Algorithms for Circuits

- MCSP $\in$ **NP**.
- But, MCSP *doesn't* seem to be **NP**-complete. (*Murray, Williams, 2017*)

Theorem (Kabanets, Cai, 2000)

If MCSP $\in$ **P**, *then:*

- $\mathbf{EXP}^{\mathbf{NP}}$ *has new circuit lower bounds.*
- $\mathbf{BPP} = \mathbf{ZPP}$.
- $\text{FACTORING}_{(D)}$, GI $\in$ **BPP**.
- *No strong PRGs / PRFs.*

Algorithms for Circuits

- MCSP $\in$ **NP**.
- But, MCSP *doesn't* seem to be **NP**-complete. (*Murray, Williams, 2017*)

Theorem (Kabanets, Cai, 2000)

If MCSP $\in \mathbf{P}$, *then:*

- $\mathbf{EXP}^{\mathbf{NP}}$ *has new circuit lower bounds.*
- $\mathbf{BPP} = \mathbf{ZPP}$.
- $\text{FACTORING}_{(D)}, \text{GI} \in \mathbf{BPP}$.
- *No strong PRGs / PRFs.*

Theorem (IKW02)

If CAPP *can be computed in* $2^{n^{o(1)}}$ *time for all circuits of size n, then* $\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$.

*Hierarchies for Semantic Classes with advice

- We have argued why we can't obtain Hierarchies for semantic measures using classical diagonalization techniques. But with using *small* advice we can obtain the following results:

*Hierarchies for Semantic Classes with advice

- We have argued why we can't obtain Hierarchies for semantic measures using classical diagonalization techniques. But with using *small* advice we can obtain the following results:

Theorem ([Bar02], [GST04])

For $a, b \in \mathbb{R}$*, with* $1 \leq a < b$*:*

$$\mathbf{BPTIME}(n^a)/1 \subsetneq \mathbf{BPTIME}(n^b)/1$$

Theorem ([FST05])

For any $1 \leq a \in \mathbb{R}$ *there is a real* $b > a$ *such that:*

$$\mathbf{RTIME}(n^b)/1 \subsetneq \mathbf{RTIME}(n^a)/\log(n)^{1/2a}$$

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages.
- The definition of $\mathbf{P}_{/\mathbf{poly}}$ is merely existential, since we haven't a way to construct such an **infinite** family of circuits.
- So, may be useful to restrict or attention to families we can construct efficiently:

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages.
- The definition of $\mathbf{P}_{/\mathbf{poly}}$ is merely existential, since we haven't a way to construct such an **infinite** family of circuits.
- So, may be useful to restrict or attention to families we can construct efficiently:

Theorem (P-Uniform Families)

*A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is **P**-uniform if there is a polynomial-time T.M. that on input 1^n outputs the description of the circuit C_n.*

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages.
- The definition of $\mathbf{P}_{/\mathbf{poly}}$ is merely existential, since we haven't a way to construct such an **infinite** family of circuits.
- So, may be useful to restrict or attention to families we can construct efficiently:

Theorem (P-Uniform Families)

*A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is **P**-uniform if there is a polynomial-time T.M. that on input 1^n outputs the description of the circuit C_n.*

Theorem

*A language L is computable by a **P**-uniform circuit family iff $L \in \mathbf{P}$.*

Uniform Families of Circuits

- We saw that $\mathbf{P}_{/\mathbf{poly}}$ contains undecidable languages.
- The definition of $\mathbf{P}_{/\mathbf{poly}}$ is merely existential, since we haven't a way to construct such an **infinite** family of circuits.
- So, may be useful to restrict or attention to families we can construct efficiently:

Theorem (P-Uniform Families)

A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is $\mathbf{P}$-uniform if there is a polynomial-time T.M. that on input 1^n outputs the description of the circuit C_n.

Theorem

A language L is computable by a $\mathbf{P}$-uniform circuit family iff $L \in \mathbf{P}$.

- We can define in the same way *logspace-uniform* circuit families, constructed by logspace-TMs.

Parallel Computations

- Circuits are a useful model for **parallel computations**.
- Number of processors $\sim$ Circuit Size
  Parallel time $\sim$ Circuit Depth

Parallel Computations

- Circuits are a useful model for **parallel computations**.
- Number of processors $\sim$ Circuit Size
  Parallel time $\sim$ Circuit Depth
- We define *logspace-uniform* circuit classes. In the same way, we can define **P**-uniform or *non-uniform* classes.

Parallel Computations

- Circuits are a useful model for **parallel computations**.
- Number of processors $\sim$ Circuit Size
  Parallel time $\sim$ Circuit Depth
- We define *logspace-uniform* circuit classes. In the same way, we can define **P**-uniform or *non-uniform* classes.

Definition (Class NC)

A language L is in $\mathbf{NC}^i$ if L is decided by a *logspace-uniform* circuit family $\{C_n\}_{n\in\mathbb{N}}$, where C_n has gates with fan-in 2, $poly(n)$ size and $\mathcal{O}\left(\log^i n\right)$ depth.

$$\mathbf{NC} = \bigcup_{i\in\mathbb{N}} \mathbf{NC}^i$$

Parallel Computations

Definition (Class AC)

A language L is in $\mathbf{AC}^i$ if L is decided by a *logspace-uniform* circuit family $\{C_n\}_{n\in\mathbb{N}}$, where C_n has gates with unbounded fan-in, $poly(n)$ size and $\mathcal{O}\left(\log^i n\right)$ depth.

$$\mathbf{AC} = \bigcup_{i\in\mathbb{N}} \mathbf{AC}^i$$

Parallel Computations

Definition (Class AC)

A language L is in $\mathbf{AC}^i$ if L is decided by a *logspace-uniform* circuit family $\{C_n\}_{n\in\mathbb{N}}$, where C_n has gates with unbounded fan-in, $poly(n)$ size and $\mathcal{O}\left(\log^i n\right)$ depth.

$$\mathbf{AC} = \bigcup_{i\in\mathbb{N}} \mathbf{AC}^i$$

- $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$, for all $i \geq 0$
- $\mathbf{NC} \subseteq \mathbf{P}$
- $\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2$
- $\mathbf{NC}^i \subseteq \mathbf{DSPACE}[\log^i n]$, for all $i \geq 0$

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous ”**P** vs **NP**” problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous "**P** vs **NP**" problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

Theorem (Shannon, 1949)

For every $n > 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit C of size $2^n/(10n)$.

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous "**P** vs **NP**" problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

Theorem (Shannon, 1949)

For every $n > 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit C of size $2^n/(10n)$.

- But after decades of efforts, the best lower bound for an **NP** language is $5n - o(n)$ (2005).

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous "**P** vs **NP**" problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

Theorem (Shannon, 1949)

For every $n > 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit C of size $2^n/(10n)$.

- But after decades of efforts, the best lower bound for an **NP** language is $5n - o(n)$ (2005).
- There are better lower bounds for some special cases (restricted classes of circuits): *bounded depth* circuits, *monotone* circuits, and bounded depth circuits with "*counting*" gates.

Boolean Functions

- A boolean function is **symmetric** if it depends only on the number of 1's in the input, and not on their positions. There are only 2^{n+1} symmetric functions out of the 2^{2^n} boolean functions.

Boolean Functions

- A boolean function is **symmetric** if it depends only on the number of 1's in the input, and not on their positions. There are only 2^{n+1} symmetric functions out of the 2^{2^n} boolean functions.

Example

- Threshold function: $THR_k(x_1, \dots, x_n) = 1$ iff $x_1 + \cdots + x_n \geq k$
- Majority function: $MAJ(x_1, \dots, x_n) = 1$ iff $x_1 + \cdots + x_n \geq \lceil n/2 \rceil$
- Parity function: $PAR(x_1, \dots, x_n) = 1$ iff $x_1 + \cdots + x_n \equiv 1 \pmod 2$
- Modular function: $MOD_k(x_1, \dots, x_n) = 1$ iff $x_1 + \cdots + x_n \equiv 0 \pmod k$

Boolean Functions

- We can encode **graph-theoretic properties** using boolean functions.
- Consider $f : \{0,1\}^{\binom{n}{2}} \rightarrow \{0,1\}$.

Boolean Functions

- We can encode **graph-theoretic properties** using boolean functions.
- Consider $f : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$.
- We associate every input variable with an edge of a n-vertices graph G.

Boolean Functions

- We can encode **graph-theoretic properties** using boolean functions.
- Consider $f : \{0,1\}^{\binom{n}{2}} \rightarrow \{0,1\}$.
- We associate every input variable with an edge of a n-vertices graph G.

Example

- Does the given graph contain at least $\binom{k}{2}$ edges?
- Does the given graph contain a clique with $\binom{k}{2}$ edges?

Boolean Functions

- We can encode **graph-theoretic properties** using boolean functions.
- Consider $f : \{0,1\}^{\binom{n}{2}} \rightarrow \{0,1\}$.
- We associate every input variable with an edge of a n-vertices graph G.

Example

- Does the given graph contain at least $\binom{k}{2}$ edges?
- Does the given graph contain a clique with $\binom{k}{2}$ edges?

- Let $CLIQUE_{k,n} : \{0,1\}^{\binom{n}{2}} \rightarrow \{0,1\}$, s.t. $CLIQUE_{k,n} = 1$ iff the encoded graph has a k-clique.

An essential lower bound: Kannan's Theorem

Theorem (Kannan's Theorem)

For every $k \in \mathbb{N}$, there is a language in $\Sigma_4^p \cap \Pi_4^p$ that is not in $\mathbf{SIZE}[n^k]$.

An essential lower bound: Kannan's Theorem

Theorem (Kannan's Theorem)

For every $k \in \mathbb{N}$*, there is a language in* $\Sigma_4^p \cap \Pi_4^p$ *that is not in* $\mathbf{SIZE}[n^k]$.

Proof:

- Let $k \in \mathbb{N}$.
- For every n, let C_n be the (*lexicographically*) first circuit such that C_n **cannot** be computed by any circuit of size at most n^k.
- By the Hierarchy Theorem, we know that such a circuit exists.
- So, if L is decided by $\{C_n\}_{n\in\mathbb{N}}$, then $L \notin \mathbf{SIZE}[n^k]$.
- We claim that $L \in \Sigma_4^p$. We need to ensure that:
  - C cannot be computed in $\mathbf{SIZE}[n^k]$.
  - C is the minimum circuit (in $\leq^{\text{lex}}$-ordering) with that property.

An essential lower bound: Kannan's Theorem

Proof (*cont'd*):

- $x \in L$ iff:
  - $\exists C \in \textbf{SIZE}[n^{k+1}]$ such that
  - $\forall C' \in \textbf{SIZE}[n^k]$
  - $\forall D,\ \langle D \rangle \leq^{\text{lex}} \langle C \rangle$
  - $\exists x' \in \{0,1\}^n :\ C(x') \neq C(x)$.
  - $\exists D' \in \textbf{SIZE}[n^k]$ such that
  - $\forall y \in \{0,1\}^n :\ D(y) = D'(y)$:
  - $C(x) = 1$.
- We need 4 alternations of quantifiers starting with $\exists$, hence $L \in \Sigma_4^p$.
- By flipping the predicate we prove also that $\overline{L} \in \Sigma_4^p$. $\square$

An essential lower bound: Kannan's Theorem

Corollary

For every $k \in \mathbb{N}$, there is a language in $\Sigma_2^p \cap \Pi_2^p$ that is not in $\mathbf{SIZE}[n^k]$.

An essential lower bound: Kannan's Theorem

Corollary

For every $k \in \mathbb{N}$, there is a language in $\Sigma_2^p \cap \Pi_2^p$ that is not in $\mathbf{SIZE}[n^k]$.

Proof (*cont'd*):

- Consider the two cases:
- If SAT $\notin \mathbf{SIZE}[n^k]$, then we 're done, since SAT $\in \mathbf{NP}$.
- If SAT $\in \mathbf{SIZE}[n^k]$, that is if $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, then by Karp-Lipton Theorem we have that $\Sigma_4^p = \Sigma_2^p$, and we have the desired language by Kannan's Theorem. $\square$

Lower Bound Techniques

- During the quest for Lower Bounds, two powerful methods were developed:

Lower Bound Techniques

- During the quest for Lower Bounds, two powerful methods were developed:
  - **Random Restrictions method**, applied to bounded depth circuits. One tries to “simplify” the circuit by *depth reduction*. Then, the resulting circuit can’t compute certain functions.

Lower Bound Techniques

- During the quest for Lower Bounds, two powerful methods were developed:
  - **Random Restrictions method**, applied to bounded depth circuits. One tries to "simplify" the circuit by *depth reduction*. Then, the resulting circuit can't compute certain functions.
  - **Polynomial Approximation Method**, where certain circuits are represented as *low-degree* polynomials (probabilistic representation). But, certain Boolean functions cannot be approximated by such polynomials.

Lower Bound Techniques

- During the quest for Lower Bounds, two powerful methods were developed:
    - **Random Restrictions method**, applied to bounded depth circuits. One tries to "simplify" the circuit by *depth reduction*. Then, the resulting circuit can't compute certain functions.
    - **Polynomial Approximation Method**, where certain circuits are represented as *low-degree* polynomials (probabilistic representation). But, certain Boolean functions cannot be approximated by such polynomials.

Reminder

Let $PAR : \{0,1\}^n \rightarrow \{0,1\}$ be the *parity* function, which outputs the modulo 2 sum of an n-bit input. That is:

$$PAR(x_1, \dots, x_n) \equiv \sum_{i=1}^{n} x_i \pmod 2$$

Lower Bound Techniques

- By using the Random Restrictions method, the following lower bound can be proved:

Theorem (Furst, Saxe, Sipser, Ajtai)

$$\mathrm{PAR} \notin \mathbf{AC}^0$$

Lower Bound Techniques

- By using the Random Restrictions method, the following lower bound can be proved:

Theorem (Furst, Saxe, Sipser, Ajtai)

$$\mathrm{PAR} \notin \mathbf{AC}^0$$

- The above result (improved by Håstad and Yao) gives a relatively tight lower bound of $\exp\left(\Omega(n^{1/(d-1)})\right)$, on the size of n-input *PAR* circuits of depth d.

Lower Bound Techniques

- By using the Random Restrictions method, the following lower bound can be proved:

Theorem (Furst, Saxe, Sipser, Ajtai)

$$\mathrm{PAR} \notin \mathbf{AC}^0$$

- The above result (improved by Håstad and Yao) gives a relatively tight lower bound of $\exp\left(\Omega(n^{1/(d-1)})\right)$, on the size of n-input *PAR* circuits of depth d.

Corollary

$$\mathbf{NC}^0 \subsetneq \mathbf{AC}^0 \subsetneq \mathbf{NC}^1$$

Random Restrictions Method

- In order to prove lower bounds for circuits of certain classes, we have to obtain a "standard form" for each circuit:
- **Standard form of a circuit C**:
  1. Push all NOT gates to the bottom layer (according to De Morgan's Laws).
  2. Each layer has the same type of gates, and adjacent layers have different types of gates.
  3. Each layer's inputs are outputs of the previous layer.
- We can easily see that every circuit (e.g. in $\mathbf{AC}^0$) can be transformed to this standard form.

Switching Lemma

Definition (Random Restriction)

A **p-random restriction ρ** is a mapping from $\{x_1, \ldots, x_n\}$ to $\{0, 1, \star\}$ applied to the Boolean function f, and the result is a function $f|_\rho$, where its variables are set according to ρ, and $\rho(x_i) = \star$ means that the variable x_i is left unassigned. Each x_i takes a value in $\{0, 1, \star\}$ with probabilities:

$$\Pr_\rho[\rho(x_i) = \star] = p$$

$$\Pr_\rho[\rho(x_i) = 0] = \Pr_\rho[\rho(x_i) = 1] = \frac{1-p}{2}$$

Switching Lemma

Theorem (Håstad's Switching Lemma)

Let f be a Boolean function that can be written as a t-DNF, and ρ a p-random restriction. Then, for any integer s:

$$\Pr_{\rho}\left[f|_{\rho} \textit{ is not an } s\text{-}CNF\right] \leq (8pt)^s$$

Switching Lemma

Theorem (Håstad's Switching Lemma)

Let f be a Boolean function that can be written as a t-DNF, and ρ a p-random restriction. Then, for any integer s:

$$\Pr_{\rho}\left[f|_{\rho} \textit{ is not an } s\text{-}CNF\right] \leq (8pt)^{s}$$

Proof Sketch (Razborov):

- Let R_ℓ denote the set of *restrictions* on n variables, leaving ℓ variables unassigned, for $1 \leq \ell \leq n$.
- $|R_\ell| = \binom{n}{\ell} 2^{n-\ell}$

Switching Lemma

Theorem (Håstad's Switching Lemma)

Let f be a Boolean function that can be written as a t-DNF, and ρ a p-random restriction. Then, for any integer s:

$$\Pr_{\rho}\left[f|_{\rho} \text{ is not an } s\text{-CNF}\right] \leq (8pt)^{s}$$

Proof Sketch (Razborov):

- Let R_ℓ denote the set of *restrictions* on n variables, leaving ℓ variables unassigned, for $1 \leq \ell \leq n$.
- $|R_\ell| = \binom{n}{\ell} 2^{n-\ell}$
- Let B be the set of **bad restrictions**, that is:

$$B(\ell, s) = \{\rho \in R_\ell \mid \text{ is } \textit{not} \text{ an } s\text{-CNF}\}$$

Switching Lemma

Proof Sketch (*cont'd*):

Lemma

For a t-DNF, then $|B(\ell, s)| \leq |R^{\ell-s}| \cdot (2t)^s$.

- We can prove the above lemma by constructing and injective function from $B(\ell, s)$ to $R^{\ell-s} \times \{0,1\}^h$, where $h = \mathcal{O}(s \log t)$.

Switching Lemma

Proof Sketch (*cont'd*):

Lemma

For a t-DNF, then $|B(\ell, s)| \leq |R^{\ell-s}| \cdot (2t)^s$.

- We can prove the above lemma by constructing and injective function from $B(\ell, s)$ to $R^{\ell-s} \times \{0,1\}^h$, where $h = \mathcal{O}(s \log t)$.
- Then,

$$\frac{|B(\ell, s)|}{|R_\ell|} \leq \frac{\binom{n}{\ell-s} 2^{n-\ell+s}(2t)^s}{\binom{n}{\ell} 2^{n-\ell}} \leq \left(\frac{\ell}{n-\ell}\right)^s (4t)^s \leq (8pt)^s$$

for $\ell = pn$ and $p \leq 1/2$. □

Switching Lemma

- Using the Switching Lemma we can prove that PAR $\notin \mathbf{AC}^0$:

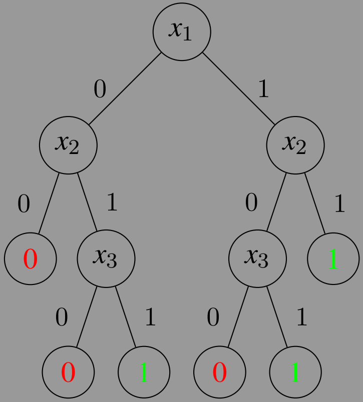
Switching Lemma

- Using the Switching Lemma we can prove that PAR $\notin$ $\mathbf{AC}^0$:
    - Let C an $\mathbf{AC}^0$ circuit, with a polynomial bound on the number of gates, and constant depth.
    - We randomly restrict more and more variables, and each step will reduce the depth by 1 (since we merge two levels with the same type of gates).
    - We take the union bound on every gate of a layer.
    - After a constant number of steps, we will have a depth 2 circuit (i.e. a k-DNF or k-CNF).
    - Such a formula can be made constant by fixing at most k of the variables.
    - But PAR is not constant under any restriction of less than n variables, so is not in $\mathbf{AC}^0$.

*Decision Trees

- **Decision Trees** are natural computational models for *boolean functions*.
- For a function $f : \{0,1\}^n \to \{0,1\}$, it is a binary tree.
- The internal nodes have labels $x_1, \ldots, x_n$, and each x_i queries the i-th bit of the input.
- After querying the variable, descend the tree light or left, depending on the value.
- The leaves have values from $\{0,1\}$, and is the value of the function in the input path.

Example



Decision Tree for $MAJ(x_1, x_2, x_3)$

*Decision Trees

Definition (Decision Tree Complexity)

The cost of a tree T on input x, denoted by $cost(T, x)$ is the number of *bits* of x examined by T. The Decision Tree Complexity of a Boolean function f is:

$$DT(f) = \min_{T \in \mathcal{T}_f} \max_{x \in \{0,1\}^n} cost(T, x)$$

where $\mathcal{T}_f$ is the set of all decision trees computing f.

- Obviously, $DT(f) \leq n$ for every $f : \{0,1\}^n \rightarrow \{0,1\}$.

*Decision Trees

Definition (Decision Tree Complexity)

The cost of a tree T on input x, denoted by $cost(T,x)$ is the number of *bits* of x examined by T. The Decision Tree Complexity of a Boolean function f is:

$$DT(f) = \min_{T \in \mathcal{T}_f} \max_{x \in \{0,1\}^n} cost(T,x)$$

where $\mathcal{T}_f$ is the set of all decision trees computing f.

- Obviously, $DT(f) \leq n$ for every $f : \{0,1\}^n \to \{0,1\}$.

Theorem (implied by Håstad's Switching Lemma)

Let f be a Boolean function that can be written as a t-DNF, and ρ a p-random restriction. Then, for any integer s:

$$\Pr_{\rho}\left[DT(f|_{\rho}) > s\right] \leq (8pt)^s$$

Circuits with counting gates

Definition

A language L is in $\mathbf{ACC}^0[m_1, \dots, m_k]$ if there is a circuit family $\{C_n\}_{n \in \mathbb{N}}$ where C_n has gates with unbounded fan-in, $poly(n)$ size and $\mathcal{O}(1)$ depth, and $MOD_{m_1}, \dots, MOD_{m_k}$ gates accepting L.

$$\mathbf{ACC}^0 = \bigcup_{m_1,\dots,m_k} \mathbf{ACC}^0[m_1, \dots, m_k]$$

- A MOD_m gate outputs 0 if the sum of its inputs is 0 $(mod m)$, and 1 otherwise.

Circuits with counting gates

Definition

A language L is in $\mathbf{ACC}^0[m_1, \dots, m_k]$ if there is a circuit family $\{C_n\}_{n\in\mathbb{N}}$ where C_n has gates with unbounded fan-in, $poly(n)$ size and $\mathcal{O}(1)$ depth, and $MOD_{m_1}, \dots, MOD_{m_k}$ gates accepting L.

$$\mathbf{ACC}^0 = \bigcup_{m_1,\dots,m_k} \mathbf{ACC}^0[m_1, \dots, m_k]$$

- A MOD_m gate outputs 0 if the sum of its inputs is 0 $(mod m)$, and 1 otherwise.

Theorem (Razborov-Smolensky,1987)

For district primes p and q, the function MOD_p is not in $\mathbf{ACC}^0[q]$.

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

- Recently, breakthrough lower bounds for **NEXP** were proved.
- Surprisingly, the lower bounds tradeoff were connected to certain algorithmic improvements.

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

- Recently, breakthrough lower bounds for **NEXP** were proved.
- Surprisingly, the lower bounds tradeoff were connected to certain algorithmic improvements.
- Let $\mathcal{C}$ a "usual" circuit class (like $\mathbf{P}_{/\mathbf{poly}}$, $\mathbf{AC}^0$ etc.)
- Define $\mathcal{C}$-SAT the circuit satisfiability problem for the class $\mathcal{C}$:

Definition ($\mathcal{C}$-SAT)

Given a circuit C_n from class $\mathcal{C}$, is there a $x \in \{0,1\}^n$ such that $C(x) = 1$?

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

- Recently, breakthrough lower bounds for **NEXP** were proved.
- Surprisingly, the lower bounds tradeoff were connected to certain algorithmic improvements.
- Let $\mathcal{C}$ a “usual” circuit class (like $\mathbf{P}_{/\mathbf{poly}}$, $\mathbf{AC}^0$ etc.)
- Define $\mathcal{C}$-SAT the circuit satisfiability problem for the class $\mathcal{C}$:

Definition ($\mathcal{C}$-SAT)

Given a circuit C_n from class $\mathcal{C}$, is there a $x \in \{0,1\}^n$ such that $C(x) = 1$?

- The trivial algorithm checks all inputs in $\mathcal{O}\left(2^n \cdot poly(n)\right)$ time.
- If we can improve this algorithm, then we can use it to construct a Boolean function in **NEXP** which has not $\mathcal{C}$-circuits.

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

- Recently, breakthrough lower bounds for **NEXP** were proved.
- Surprisingly, the lower bounds tradeoff were connected to certain algorithmic improvements.
- Let $\mathcal{C}$ a "usual" circuit class (like $\mathbf{P}_{/\mathbf{poly}}$, $\mathbf{AC}^0$ etc.)
- Define $\mathcal{C}$-SAT the circuit satisfiability problem for the class $\mathcal{C}$:

Definition ($\mathcal{C}$-SAT)

Given a circuit C_n from class $\mathcal{C}$, is there a $x \in \{0,1\}^n$ such that $C(x) = 1$?

- The trivial algorithm checks all inputs in $\mathcal{O}\left(2^n \cdot poly(n)\right)$ time.
- If we can improve this algorithm, then we can use it to construct a Boolean function in **NEXP** which has not $\mathcal{C}$-circuits.
- Hence:

$$\boxed{\text{Better algorithm for } \mathcal{C}\text{-SAT} \longrightarrow \mathbf{NEXP} \not\subseteq \mathcal{C}}$$

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

Theorem (Williams, 2010)

Let $s(n)$ be a superpolynomial function. If CIRCUIT SAT *on n inputs and $poly(n)$ size can be solved in $2^n \cdot poly(n)/s(n)$, then:*

$$\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$$

- We can substitute $\mathbf{P}_{/\mathbf{poly}}$ with any other "usual" circuit class.

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

Theorem (Williams, 2010)

Let $s(n)$ be a superpolynomial function. If CIRCUIT SAT *on n inputs and $poly(n)$ size can be solved in $2^n \cdot poly(n)/s(n)$, then:*

$$\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$$

- We can substitute $\mathbf{P}_{/\mathbf{poly}}$ with any other “usual” circuit class.
- But, for circuits in $\mathbf{ACC}^0$ there are advacements. The work of Yao, Beigel and Tarui showed that brute force can be beaten for $\mathbf{ACC}^0$-SAT. Hence:

Lower Bounds for **NEXP**: Algorithms vs Lower Bounds

Theorem (Williams, 2010)

Let $s(n)$ be a superpolynomial function. If CIRCUIT SAT *on n inputs and poly(n) size can be solved in $2^n \cdot poly(n)/s(n)$, then:*

$$\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$$

- We can substitute $\mathbf{P}_{/\mathbf{poly}}$ with any other "usual" circuit class.
- But, for circuits in $\mathbf{ACC}^0$ there are advacements. The work of Yao, Beigel and Tarui showed that brute force can be beaten for $\mathbf{ACC}^0$-SAT. Hence:

Theorem (Williams, 2011)

$$\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$$

*We will later see a sketch of Williams' proof (after discussing the Easy Witness Lemma)

Monotone Circuits

Definition

For $x, y \in \{0,1\}^n$, we denote $x \preceq y$ if every bit that is 1 in x is also 1 in y. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $f(x) \leq f(y)$ for every $x \preceq y$.

Definition

A Boolean Circuit is *monotone* if it contains only AND and OR gates, and no NOT gates. Such a circuit can only compute monotone functions.

Monotone Circuits

Definition

For $x, y \in \{0,1\}^n$, we denote $x \preceq y$ if every bit that is 1 in x is also 1 in y. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $f(x) \leq f(y)$ for every $x \preceq y$.

Definition

A Boolean Circuit is *monotone* if it contains only AND and OR gates, and no NOT gates. Such a circuit can only compute monotone functions.

Theorem (Razborov, Andreev, Alon, Boppana)

There exists some constant $\epsilon > 0$ such that for every $k \leq n^{1/4}$, there is no monotone circuit of size less than $2^{\epsilon\sqrt{k}}$ that computes $CLIQUE_{k,n}$.

- This is a significant lower bound ($2^{\Omega(n^{1/8})}$), but...

Natural Proofs

- Where is the problem finally?

Natural Proofs

- Where is the problem finally?
- Today, we know *that a result for a lower bound using such techniques would imply the inversion of strong one-way functions:*

Natural Proofs

- Where is the problem finally?
- Today, we know *that a result for a lower bound using such techniques would imply the inversion of strong one-way functions:*

Definition (Razborov, Rudich 1994)

Let $\mathcal{P}$ be the predicate:

> *"A Boolean function $f : \{0,1\}^n \to \{0,1\}$ doesn't have n^c-sized circuits for some $c \geq 1$."*

$\mathcal{P}(f) = 0, \forall f \in \mathbf{SIZE}(n^c)$ for a $c \geq 1$. We call this ***n^c-usefulness***.
A predicate $\mathcal{P}$ is natural if:

- There is an algorithm $M \in \mathbf{E}$ such that for a function $g : \{0,1\}^n \to \{0,1\}$: $M(g) = \mathcal{P}(g)$ (**Constructiveness**)
- For a random function g: $\mathbf{Pr}[\mathcal{P}(g) = 1] \geq \frac{1}{n}$ (**Largeness**)

Natural Proofs

Theorem

If strong one-way functions exist, then there exists a constant $c \in \mathbb{N}$ such that there is no n^c-useful natural predicate $\mathcal{P}$.

Natural Proofs

Theorem

If strong one-way functions exist, then there exists a constant $c \in \mathbb{N}$ such that there is no n^c-useful natural predicate $\mathcal{P}$.

Example

Håstad's Switching Lemma defines the property:
$\mathcal{P}(f) = 1$ iff f cannot be made constant by fixing a portion of the variables.

- The property is **useful** against $\mathbf{AC}^0$.
- The property is **constructive** in $\mathbf{E}$ by enumerating all restrictions and checking the inputs.
- Also, the property satisfies the **largeness** condition, by calculating the (negligible) fraction of Boolean functions that can be made constant under restrictions.

Natural Proofs

- Recently, it was shown that **constructivity** is unavoidable:

Theorem (Williams, 2013)

NEXP $\not\subseteq \mathcal{C}$ *is equivalent to exhibiting a constructive property that is useful against* $\mathcal{C}$.

*Algorithms from Circuit Lower Bounds

- We saw that better algorithms for $\mathcal{C}$-SAT imply new lower bounds.
- Is the opposite possible? Can lower bound techniques be used to derive new algorithms?

*Algorithms from Circuit Lower Bounds

- We saw that better algorithms for $\mathcal{C}$-SAT imply new lower bounds.
- Is the opposite possible? Can lower bound techniques be used to derive new algorithms?
- Recall the problem APSP (All-pairs shortest paths):
- The classic DP algorithm (Floyd-Washall) solves it in $\mathcal{O}\left(n^3\right)$, where n the number of graph's vertices.

*Algorithms from Circuit Lower Bounds

- We saw that better algorithms for $\mathcal{C}$-SAT imply new lower bounds.
- Is the opposite possible? Can lower bound techniques be used to derive new algorithms?
- Recall the problem APSP (All-pairs shortest paths):
- The classic DP algorithm (Floyd-Washall) solves it in $\mathcal{O}\left(n^3\right)$, where n the number of graph's vertices.
- By using the Razborov-Smolensky's polynomial approximation method, the following holds:

Theorem (Williams, 2016)

The All-Pairs Shortest Paths problem can be solved in time:

$$\frac{n^3}{2^{\Omega(\sqrt{\log n})}}$$

*Algorithms from Circuit Lower Bounds

- Another significant problem is **Orthogonal Vectors** (OV):

Definition (OV)

Given two sets of vectors $A, B \subseteq \{0,1\}^d$, $|A| = |B| = n$, are there $x \in A$ and $y \in B$ such that:

$$x \cdot y = \sum_{i \in [d]} x_i \cdot y_i = 0\ ?$$

*Algorithms from Circuit Lower Bounds

- Another significant problem is **Orthogonal Vectors** (OV):

Definition (OV)

Given two sets of vectors $A, B \subseteq \{0,1\}^d$, $|A| = |B| = n$, are there $x \in A$ and $y \in B$ such that:

$$x \cdot y = \sum_{i \in [d]} x_i \cdot y_i = 0\ ?$$

- The naïve algorithm solves the problem in $\mathcal{O}\left(n^2 d\right)$ time.

*Algorithms from Circuit Lower Bounds

- Another significant problem is **Orthogonal Vectors** (OV):

Definition (OV)

Given two sets of vectors $A, B \subseteq \{0,1\}^d$, $|A| = |B| = n$, are there $x \in A$ and $y \in B$ such that:

$$x \cdot y = \sum_{i \in [d]} x_i \cdot y_i = 0 \, ?$$

- The naïve algorithm solves the problem in $\mathcal{O}\left(n^2 d\right)$ time.

Theorem (Williams, 2016)

The Orthogonal Vectors problem can be solved in time:

$$n^{2 - \frac{1}{O(\log \frac{d}{\log n})}}$$

Summary 1/2

- In non-uniform complexity, we allow the program size to grow along with the input.
- $\mathbf{P}_{/\mathbf{poly}}$, the class of languages having polynomial-sized circuit families, is the non-uniform analogue of $\mathbf{P}$.
- $\mathbf{P}_{/\mathbf{poly}}$ can be equivalently defined as the class of polynomial-time TMs with *polynomial advice*.
- $\mathbf{P}$ and $\mathbf{BPP}$ are contained in $\mathbf{P}_{/\mathbf{poly}}$.
- If $\mathbf{NP} \subset \mathbf{P}_{/\mathbf{poly}}$, then $\mathbf{PH} = \Sigma_2^p$.
- If $\mathbf{EXP} \subset \mathbf{P}_{/\mathbf{poly}}$, then $\mathbf{EXP} = \Sigma_2^p$.

Summary 2/2

- Most Boolean functions require exponential-size circuits.
- If we find an **NP** language which doesn't have polynomial-size circuits, then $\mathbf{P} \neq \mathbf{NP}$.
- The Parity function is *not* in $\mathbf{AC}^0$.
- Algorithmic improvements can imply circuit lower bounds.
- The Natural Proofs barrier indicate that common lower bound proof techniques do not suffice for proving the desired lower bounds.

Contents

Introduction

"Maybe Fermat had a proof! But an important party was certainly missing to make the proof complete: the verifier. Each time rumor gets around that a student somewhere proved **P** $=$ **NP**, *people ask "Has Karp seen the proof?" (they hardly even ask the student's name). Perhaps the verifier is most important that the prover." (from [BM88])*

- The notion of a mathematical proof is related to the certificate definition of **NP**.
- We enrich this scenario by introducing **interaction** in the basic scheme:
  The person (or TM) who verifies the proof asks the person who provides the proof a series of "queries", before he is convinced, and if he is, he provide the certificate.

Introduction

- The first person will be called **Verifier**, and the second **Prover**.
- In our model of computation, Prover and Verifier are interacting Turing Machines.
- We will categorize the various proof systems created by using:
  - various TMs (nondeterministic, probabilistic etc)
  - the information exchanged (private/public coins etc)
  - the number of TMs (IPs, MIPs,...)

Warmup: Interactive Proofs with deterministic Verifier

Definition (Deterministic Proof Systems)

We say that a language L has a k-round deterministic interactive proof system if there is a deterministic Turing Machine V that on input $x, \alpha_1, \alpha_2, \ldots, \alpha_i$ runs in time polynomial in $|x|$, and can have a k-round interaction with any TM P such that:

- $x \in L \Rightarrow \exists P : \langle V, P \rangle(x) = 1$ (*Completeness*)
- $x \notin L \Rightarrow \forall P : \langle V, P \rangle(x) = 0$ (*Soundness*)

The class **dIP** contains all languages that have a k-round deterministic interactive proof system, where p is polynomial in the input length.

- $\langle V, P \rangle(x)$ denotes the output of V at the end of the interaction with P on input x, and α_i the exchanged strings.
- The above definition does not place limits on the computational power of the Prover!

Warmup: Interactive Proofs with deterministic Verifier

- But...

Theorem

$$\mathbf{dIP} = \mathbf{NP}$$

Proof: Trivially, $\mathbf{NP} \subseteq \mathbf{dIP}$. ✓
Let $L \in \mathbf{dIP}$:

- A certificate is a transcript $(\alpha_1, \ldots, \alpha_k)$ causing V to accept, i.e. $V(x, \alpha_1, \ldots, \alpha_k) = 1$.
- We can efficiently check if $V(x) = \alpha_1$, $V(x, \alpha_1, \alpha_2) = \alpha_3$ etc...
  - If $x \in L$ such a transcript exists!
  - Conversely, if a transcript exists, we can define define a proper P to satisfy: $P(x, \alpha_1) = \alpha_2$, $P(x, \alpha_1, \alpha_2, \alpha_3) = \alpha_4$ etc., so that $\langle V, P\rangle(x) = 1$, so $x \in L$.
- So $L \in \mathbf{NP}$! $\square$

Probabilistic Verifier: The Class IP

- We saw that if the verifier is a simple deterministic TM, then the interactive proof system is described precisely by the class **NP**.
- Now, we let the *verifier* be probabilistic, i.e. the verifier's queries will be computed using a probabilistic TM:

Definition (Goldwasser-Micali-Rackoff)

For an integer $k \geq 1$ (that may depend on the input length), a language L is in $\mathbf{IP}[k]$ if there is a probabilistic polynomial-time T.M. V that can have a k-round interaction with a T.M. P such that:

- $x \in L \Rightarrow \exists P : Pr[\langle V, P \rangle(x) = 1] \geq \frac{2}{3}$ (*Completeness*)
- $x \notin L \Rightarrow \forall P : Pr[\langle V, P \rangle(x) = 1] \leq \frac{1}{3}$ (*Soundness*)

Probabilistic Verifier: The Class IP

Definition

We also define:

$$\mathbf{IP} = \bigcup_{c \in \mathbb{N}} \mathbf{IP}[n^c]$$

- The "output" $\langle V, P \rangle(x)$ is a random variable.
- We'll see that **IP** is a very large class! ($\supseteq$ **PH**)
- As usual, we can replace the completeness parameter $2/3$ with $1 - 2^{-n^s}$ and the soundness parameter $1/3$ by 2^{-n^s}, without changing the class for any fixed constant $s > 0$.
- We can also replace the completeness constant $2/3$ with 1 (**perfect completeness**), without changing the class, but replacing the soundness constant $1/3$ with 0, is equivalent with a *deterministic verifier*, so class **IP** collapses to **NP**.

Interactive Proof for Graph Non-Isomorphism

Definition

Two graphs G_1 and G_2 are *isomorphic*, if there exists a permutation π of the labels of the nodes of G_1, such that $\pi(G_1) = G_2$. If G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$.

- GI: Given two graphs G_1, G_2, decide if they are isomorphic.
- GNI: Given two graphs G_1, G_2, decide if they are *not* isomorphic.

- Obviously, GI $\in$ **NP** and GNI $\in$ *co***NP**.
- This proof system relies on the Verifier's access to a *private* random source which cannot be seen by the Prover, so we confirm the crucial role the private coins play.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1, 2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H. Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

Interactive Proof for Graph Non-Isomorphism

Verifier: Picks $i \in \{1, 2\}$ uniformly at random.
Then, it permutes randomly the vertices of G_i to get a new graph H. Is sends H to the Prover.
Prover: Identifies which of G_1, G_2 was used to produce H.
Let G_j be the graph. Sends j to V.
Verifier: Accept if $i = j$. Reject otherwise.

- If $G_1 \ncong G_2$, then the powerfull prover can (*nondeterministically*) guess which one of the two graphs is isomprphic to H, and so the Verifier accepts with probability 1.
- If $G_1 \cong G_2$, the prover can't distinguish the two graphs, since a random permutation of G_1 looks exactly like a random permutation of G_2. So, the best he can do is guess randomly one, and the Verifier accepts with probability (at most) 1/2, which can be reduced by additional repetitions.

Babai's Arthur-Merlin Games

Definition (Extended (FGMSZ89))

An Arhur-Merlin Game is a pair of interactive TMs A and M, and a predicate R such that:

- On input x, exactly $2q(|x|)$ messages of length $m(|x|)$ are exchanged, $q, m \in poly(|x|)$.
- A goes first, and at iteration $1 \leq i \leq q(|x|)$ chooses u.a.r. a string r_i of length $m(|x|)$.
- M's reply in the i^{th} iteration is $y_i = M(x, r_1, \dots, r_i)$ (M's strategy).
- For every M', a **conversation** between A and M' on input x is $r_1 y_1 r_2 y_2 \cdots r_{q(|x|)} y_{q(|x|)}$.
- The set of all conversations is denoted by $CONV_x^{M'}$, $|CONV_x^{M'}| = 2^{q(|x|)m(|x|)}$.

Babai's Arthur-Merlin Games

Definition (*cont'd*)

- The predicate R maps the input x and a conversation to a Boolean value.
- The set of accepting conversations is denoted by $ACC_x^{R,M}$, and is the set:

$$\{r_1 \cdots r_q | \exists y_1 \cdots y_q \ s.t. \ r_1 y_1 \cdots r_q y_q \in CONV_x^M \wedge R(r_1 y_1 \cdots r_q y_q) = 1\}$$

- A language L has an Arthur-Merlin proof system if:
  - **There exists** a strategy for M, such that for all $x \in L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \geq \frac{2}{3}$ (*Completeness*)
  - **For every** strategy for M, and for every $x \notin L$: $\frac{ACC_x^{R,M}}{CONV_x^M} \leq \frac{1}{3}$ (*Soundness*)

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin $\rightarrow$ Prover**
- **Arthur $\rightarrow$ Verifier**

Definitions

- So, with respect to the previous **IP** definition:

Definition

For every k, the complexity class $\mathbf{AM}[k]$ is defined as a subset to $\mathbf{IP}[k]$ obtained when we restrict the verifier's messages to be *random bits*, and not allowing it to use any other random bits that are not contained in these messages.
We denote $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **Merlin $\rightarrow$ Prover**
- **Arthur $\rightarrow$ Verifier**
- Also, the class **MA** consists of all languages L, where there's an interactive proof for L in which the prover first sending a message, and then the verifier is "tossing coins" and computing its decision by doing a deterministic polynomial-time computation involving the input, the message and the random output.

Public vs. Private Coins

Theorem

$$\mathrm{GNI} \in \mathbf{AM}[2]$$

Theorem

For every $p \in poly(n)$:

$$\mathbf{IP}\,(p(n)) = \mathbf{AM}(p(n) + 2)$$

- So,

$$\mathbf{IP}[poly] = \mathbf{AM}[poly]$$

Properties of Arthur-Merlin Games

- $\mathbf{MA} \subseteq \mathbf{AM}$
- $\mathbf{MA}[1] = \mathbf{NP}$, $\mathbf{AM}[1] = \mathbf{BPP}$
- **AM** could be intuitively approached as the probabilistic version of **NP** (usually denoted as $\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$).
- $\mathbf{AM} \subseteq \Pi_2^p$ and $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$.
- $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{BPP}}$, $\mathbf{MA}^{\mathbf{BPP}} = \mathbf{MA}$, $\mathbf{AM}^{\mathbf{BPP}} = \mathbf{AM}$ and $\mathbf{AM}^{\Delta\Sigma_1^p} = \mathbf{AM}^{\mathbf{NP} \cap co\mathbf{NP}} = \mathbf{AM}$
- If we consider the complexity classes $\mathbf{AM}[k]$ (the languages that have Arthur-Merlin proof systems of a bounded number of rounds, they form an hierarchy:

$$\mathbf{AM}[0] \subseteq \mathbf{AM}[1] \subseteq \cdots \subseteq \mathbf{AM}[k] \subseteq \mathbf{AM}[k+1] \subseteq \cdots$$

- Are these inclusions proper ? ? ?

Properties of Arthur-Merlin Games

$$\begin{tikzcd}
 & \mathbf{NP} \arrow[r] & \mathbf{MA} \arrow[rr] \arrow[dr] & & \Sigma_2^p \\
 & & & \mathbf{AM} \arrow[dddr] & \\
\mathbf{P} \arrow[uur] \arrow[r] \arrow[ddr] & \mathbf{BPP} \arrow[uur] \arrow[ddr] & & & \\
 & & & co\mathbf{AM} \arrow[uuur] & \\
 & co\mathbf{NP} \arrow[r] & co\mathbf{MA} \arrow[ur] \arrow[rr] & & \Pi_2^p
\end{tikzcd}$$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x,y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x,y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Properties of Arthur-Merlin Games

Definition

We denote as $\mathcal{C} = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall, \exists^+\}$, the class $\mathcal{C}$ of languages L satisfying:

- $x \in L \Rightarrow Q_1 y\, R(x,y)$
- $x \notin L \Rightarrow Q_2 y\, \neg R(x,y)$

- So: $\mathbf{P} = (\forall/\forall)$, $\mathbf{NP} = (\exists/\forall)$, $co\mathbf{NP} = (\forall/\exists)$
  $\mathbf{BPP} = (\exists^+/\exists^+)$, $\mathbf{RP} = (\exists^+/\forall)$, $co\mathbf{RP} = (\forall/\exists^+)$

Arthur-Merlin Games

$$\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall)$$

$$\mathbf{MA} = \mathcal{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+)$$

- Similarly: $\mathbf{AMA} = (\exists^+\exists\exists^+/\exists^+\forall\exists^+)$ etc.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ (1) (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ (2)

i) $\mathbf{MA} = \mathcal{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$ (*the last inclusion holds by quantifier contraction*). Also, $(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.
ii) Similarly,
$\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also, $(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

- i $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
- ii $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathcal{N}\cdot\mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$
(*the last inclusion holds by quantifier contraction*). Also,
$(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.
ii) Similarly,
$\mathbf{AM} = \mathcal{BP}\cdot\mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also,
$(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

1. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
2. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathcal{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$
(*the last inclusion holds by quantifier contraction*). Also,
$(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.
ii) Similarly,
$\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also,
$(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

i. $\mathbf{MA} = (\exists\forall/\forall\exists^+)$
ii. $\mathbf{AM} = (\forall\exists/\exists^+\forall)$

Proof:

Lemma

- $\mathbf{BPP} = (\exists^+/\exists^+) = (\exists^+\forall/\forall\exists^+) = (\forall\exists^+/\exists^+\forall)$ **(1)** (BPP-Theorem)
- $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$ **(2)**

i) $\mathbf{MA} = \mathcal{N} \cdot \mathbf{BPP} = (\exists\exists^+/\forall\exists^+) \overset{(1)}{=} (\exists\exists^+\forall/\forall\forall\exists^+) \subseteq (\exists\forall/\forall\exists^+)$
(*the last inclusion holds by quantifier contraction*). Also,
$(\exists\forall/\forall\exists^+) \subseteq (\exists\exists^+/\forall\exists^+) = \mathbf{MA}$.
ii) Similarly,
$\mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP} = (\exists^+\exists/\exists^+\forall) = (\forall\exists^+\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall)$. Also,
$(\forall\exists/\exists^+\forall) \subseteq (\exists^+\exists/\exists^+\forall) = \mathbf{AM}$.

Properties of Arthur-Merlin Games

Theorem

$$\mathbf{MA} \subseteq \mathbf{AM}$$

Proof:
Obvious from (2): $(\exists\forall/\forall\exists^+) \subseteq (\forall\exists/\exists^+\forall)$. $\square$

Theorem

1. $\mathbf{AM} \subseteq \Pi_2^p$
2. $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$

Proof:
i) $\mathbf{AM} = (\forall\exists/\exists^+\forall) \subseteq (\forall\exists/\exists\forall) = \Pi_2^p$
ii) $\mathbf{MA} = (\exists\forall/\forall\exists^+) \subseteq (\exists\forall/\forall\exists) = \Sigma_2^p$, and
$\mathbf{MA} \subseteq \mathbf{AM} \Rightarrow \mathbf{MA} \subseteq \Pi_2^p$. So, $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$. $\square$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^+\exists/\forall\exists^+\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^+\forall\exists/\forall\forall\exists^+\forall) \subseteq (\exists\forall\exists/\forall\exists^+\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^+\forall\forall) \subseteq (\forall\exists/\exists^+\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Theorem (Speedup Theorem)

For $t(n) \geq 2$:

$$\mathbf{AM}[2t(n)] = \mathbf{AM}[t(n)]$$

- **The Arthur-Merlin Hierarchy collapses at its second level:**

Theorem (Collapse Theorem)

For every $k \geq 2$:

$$\mathbf{AM} = \mathbf{AM}[k] = \mathbf{MA}[k+1]$$

Example

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(\mathbf{1})}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(\mathbf{2})}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Properties of Arthur-Merlin Games

Proof:

- The general case is implied by the generalization of BPP-Theorem (**1**) & (**2**):
- $(\mathbf{Q_1}\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\exists^+\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\forall\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ ($\mathbf{1'}$)
- $(\mathbf{Q_1}\exists\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) \subseteq (\mathbf{Q_1}\forall\exists\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ ($\mathbf{2'}$)
- Using the above we can easily see that the Arthur-Merlin Hierarchy collapses at the second level. (*Try it!*) $\square$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if GI is* ***NP****-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \square$$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if GI is* ***NP****-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(\mathbf{2})}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \square$$

Properties of Arthur-Merlin Games

Theorem (BHZ)

*If co***NP** $\subseteq$ **AM** *(that is, if GI is* ***NP****-complete), then the Polynomial Hierarchy collapses at the second level, and* **PH** $= \Sigma_2^p =$ **AM**.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(\mathbf{2})}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \ \square$$

Properties of Arthur-Merlin Games

Theorem (BHZ)

If co$\mathbf{NP} \subseteq \mathbf{AM}$ *(that is, if GI is* $\mathbf{NP}$*-complete), then the Polynomial Hierarchy collapses at the second level, and* $\mathbf{PH} = \Sigma_2^p = \mathbf{AM}$.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$
Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(\mathbf{2})}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \square$$

Properties of Arthur-Merlin Games

Theorem (BHZ)

If co$\mathbf{NP} \subseteq \mathbf{AM}$ *(that is, if GI is* $\mathbf{NP}$*-complete), then the Polynomial Hierarchy collapses at the second level, and* $\mathbf{PH} = \Sigma_2^p = \mathbf{AM}$.

Proof: Our hypothesis states: $(\forall/\exists) \subseteq (\forall\exists/\exists^+\forall)$

Then:

$$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(\mathbf{2})}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) = \mathbf{AM} \subseteq (\forall\exists/\exists\forall) = \Pi_2^p. \square$$

Measure One Results

- $\mathbf{P}^A \neq \mathbf{NP}^A$, $\mathbf{P}^A = \mathbf{BPP}^A$, $\mathbf{NP}^A = \mathbf{AM}^A$, for almost all oracles A.

Definition

$$almost\mathcal{C} = \left\{L \middle| \mathbf{Pr}_{A\in\{0,1\}^*}\left[L \in \mathcal{C}^A\right] = 1\right\}$$

Theorem

1. $almost\boldsymbol{P} = \boldsymbol{BPP}$ [BG81]
2. $almost\boldsymbol{NP} = \boldsymbol{AM}$ [NW94]
3. $almost\boldsymbol{PH} = \boldsymbol{PH}$

Theorem (Kurtz)

For almost every pair of oracles B, C:

1. $\mathbf{BPP} = \mathbf{P}^B \cap \mathbf{P}^C$
2. $almost\boldsymbol{NP} = \mathbf{NP}^B \cap \mathbf{NP}^C$

The power of Interactive Proofs

- As we saw, **Interaction** alone does not gives us computational capabilities beyond **NP**.
- Also, **Randomization** alone does not give us significant power (we know that $\mathbf{BPP} \subseteq \Sigma_2^p$, and many researchers believe that $\mathbf{P} = \mathbf{BPP}$, which holds under some plausible assumptions).
- How much power could we get by their *combination*?
- We know that for fixed $k \in \mathbb{N}$, $\mathbf{IP}[k]$ collapses to

$$\mathbf{IP}[k] = \mathbf{AM} = \mathcal{BP} \cdot \mathbf{NP}$$

  a class that is “close” to **NP** (*under similar assumptions, the non-deterministic analogue of* ***P*** *vs.* ***BPP*** *is* ***NP*** *vs.* ***AM***.)
- If we let k be a polynomial in the size of the input, how much more power could we get?

The power of Interactive Proofs

- Surprisingly:

Theorem (L.F.K.N. & Shamir)

$$\mathbf{IP} = \mathbf{PSPACE}$$

The power of Interactive Proofs

Lemma 1

$$\mathbf{IP} \subseteq \mathbf{PSPACE}$$

The power of Interactive Proofs

Lemma 1

$$\mathbf{IP} \subseteq \mathbf{PSPACE}$$

Proof:

- If the Prover is an **NP**, or even a **PSPACE** machine, the lemma holds.
- But what if we have an omnipotent prover?
- On any input, the Prover chooses its messages in order to *maximize the probability of V's acceptance*!
- We consider the prover as an **oracle**, by assuming wlog that his responses are one bit at a time.
- The protocol has polynomially many rounds (say N=n^c), which bounds the messages and the random bits used.
- So, the protocol is described by a computation tree T:

The power of Interactive Proofs

Proof(cont'd):

- Vertices of T are V's configurations.
- **Random Branches** (queries to the random tape)
- **Oracle Branches** (queries to the prover)
- For each fixed P, the tree T_P can be pruned to obtain only random branches.
- Let $\mathbf{Pr}_{opt}[E \mid F]$ the conditional probability given that the prover *always behaves optimally*.
- The acceptance condition is $m_N = 1$.
- For $y_i \in \{0,1\}^N$ and $z_i \in \{0,1\}$ let:

$$R_i = \bigwedge_{j=1}^{i} m_j = y_j$$

$$S_i = \bigwedge_{j=1}^{i} l_j = z_j$$

The power of Interactive Proofs

Proof(cont'd):

- $$\mathbf{Pr}_{opt}[m_N = 1 \mid R_{i-1} \wedge S_{i-1}] =$$
$$\sum_{y_i} \max_{z_i} \mathbf{Pr}_{opt}[m_N = 1 \mid R_i \wedge S_i] \cdot \mathbf{Pr}_{opt}[R_i \mid R_{i-1} \wedge S_{i-1}]$$
- $\mathbf{Pr}_{opt}[R_i \mid R_{i-1} \wedge S_{i-1}]$ is **PSPACE**-computable, by simulating V.
- $\mathbf{Pr}_{opt}[m_N = 1 \mid R_i \wedge S_i]$ can be calculated by DFS on T.
- The probability of acceptance is
$\mathbf{Pr}_{opt}[m_N = 1] = \mathbf{Pr}_{opt}[m_N = 1 \mid R_0 \wedge S_0]$
- The prover can calculate its optimal move at any point in the protocol in **PSPACE** by calculating $\mathbf{Pr}_{opt}[m_N = 1 \mid R_i \wedge S_i]$ for $z_i\{0, 1\}$ and choosing its answer to be the value that gives the maximum. $\square$

Warmup: Interactive Proof for UNSAT

Lemma 2

$$\mathbf{PSPACE} \subseteq \mathbf{IP}$$

- For simplicity, we will construct an Interactive Proof for UNSAT (a *co***NP**-complete problem), showing that:

Theorem

$$co\mathbf{NP} \subseteq \mathbf{IP}$$

- Let N be a prime.
- We will translate a **formula** ϕ with m clauses and n variables $x_1, \ldots, x_n$ to a **polynomial** p over the field $(modN)$ (where $N > 2^n \cdot 3^m$), in the following way:

Arithmetization

- Arithmetic generalization of a CNF Boolean Formula.

$$\begin{array}{rcl} \mathrm{T} & \longrightarrow & 1 \\ \mathrm{F} & \longrightarrow & 0 \\ \neg x & \longrightarrow & 1-x \\ \wedge & \longrightarrow & \times \\ \vee & \longrightarrow & + \end{array}$$

Example

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee x_4)$$
$$\downarrow$$
$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + x_4)$$

- Each literal is of degree 1, so the polynomial p is of degree at most m.
- Also, $0 < p < 3^m$.

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$ picks $r_n \in \{0, \ldots, N-1\}$

Warmup: Interactive Proof for UNSAT

Prover		**Verifier**
Sends primality proof for N	$\longrightarrow$	checks proof
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	checks if $q_1(0) + q_1(1) = 0$
	$\longleftarrow$	sends $r_1 \in \{0, \ldots, N-1\}$
$q_2(x) = \sum p(r_1, x, x_3, \ldots x_n)$	$\longrightarrow$	checks if $q_2(0) + q_2(1) = q_1(r_1)$
	$\longleftarrow$	sends $r_2 \in \{0, \ldots, N-1\}$
	$\vdots$	
$q_n(x) = p(r_1, \ldots, r_{n-1}, x)$	$\longrightarrow$	checks if $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$
		picks $r_n \in \{0, \ldots, N-1\}$
		checks if $q_n(r_n) = p(r_1, \ldots, r_n)$

Warmup: Interactive Proof for UNSAT

- If ϕ is **unsatisfiable**,then

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n) \equiv 0 \ (modN)$$

  and the protocol will succeed.
- Also, the arithmetization can be done in polynomial time, and if we take $N = 2^{\mathcal{O}(n+m)}$, then the elements in the field can be represented by $\mathcal{O}(n+m)$ bits, and thus an evaluation of p in any point of $\{0, \ldots, N-1\}$ can be computed in polynomial time.
- We have to show that if ϕ is satisfiable, then the verifier will **reject** with high probability.
- If ϕ is satisfiable, then $\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n) \neq 0 \ (modN)$

- So, $p_1(0) + p_1(1) \neq 0$, so if the prover send p_1 we 're done.
- If the prover send $q_1 \neq p_1$, then the polynomials will agree on at most m places. So, $\mathbf{Pr}\left[p_1(r_1) \neq q_1(r_1)\right] \geq 1 - \frac{m}{N}$.
- If indeed $p_1(r_1) \neq q_1(r_1)$ and the prover sends $p_2 = q_2$, then the verifier will reject since $q_2(0) + q_2(1) = p_1(r_1) \neq q_1(r_1)$.
- Thus, the prover must send $q_2 \neq p_2$.
- *We continue in a similar way*: If $q_i \neq p_i$, then with probability at least $1 - \frac{m}{N}$, r_i is such that $q_i(r_i) \neq p_i(r_i)$.
- Then, the prover must send $q_{i+1} \neq p_{i+1}$ in order for the verifier not to reject.
- At the end, if the verifier has not rejected before the last check, $\mathbf{Pr}\left[p_n \neq q_n\right] \geq 1 - (n-1)\frac{m}{N}$.
- If so, with probability at least $1 - \frac{m}{N}$ the verifier will reject since, $q_n(x)$ and $p(r_1, \ldots, r_{n-1}, x)$ differ on at least that fraction of points.
- **The total probability that the verifier will accept if at most $\frac{nm}{N}$.**

Arithmetization of QBF

$$\exists \longrightarrow \sum$$
$$\forall \longrightarrow \prod$$

Example

$$\forall x_1 \exists x_2 [(x_1 \wedge x_2) \vee \exists x_3 (\bar{x}_2 \wedge x_3)]$$

$$\downarrow$$

$$\prod_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left[(x_1 \cdot x_2) + \sum_{x_3 \in \{0,1\}} (1 - x_2) \cdot x_3 \right]$$

Arithmetization of QBF

- **But**, every quantifier arithmetization may double the degree of each variable, leading to an exponential degree polynomial. The verifier can't read this.
- We can substitute the arithmetized polynomial with another, agreeing with the original only on all boolean assignments:
  - Since if $x = 0, 1$ then $x^i = x$, for all i, we can just get rid of the exponents.
- So, we can arithmetize Quantified Boolean Formulas, and with slight modifications, the same protocol works.
- Remember that the TQBF problem is **PSPACE**-complete.
- Hence, **PSPACE** $\subseteq$ **IP**. $\square$

Epilogue: Probabilistically Checkable Proofs

- But if we put a **proof** instead of a Prover?

Epilogue: Probabilistically Checkable Proofs

- But if we put a **proof** instead of a Prover?
- The alleged proof is a string, and the (probabilistic) verification procedure is given direct (**oracle**) access to the proof.
- The verification procedure can access only *few* locations in the proof!
- We parameterize these Interactive Proof Systems by two complexity measures:
  - **Query** Complexity
  - **Randomness** Complexity
- The effective proof length of a PCP system is upper-bounded by $q(n) \cdot 2^{r(n)}$ (in the non-adaptive case).

PCP Definitions

Definition (PCP Verifiers)

Let L be a language and $q, r : \mathbb{N} \to \mathbb{N}$. We say that L has an $(r(n), q(n))$-**PCP** verifier if there is a probabilistic polynomial-time algorithm V (the **verifier**) satisfying:

- *Efficiency*: On input $x \in \{0,1\}^*$ and given random oracle access to a string $\pi \in \{0,1\}^*$ of length at most $q(n) \cdot 2^{r(n)}$ (which we call the **proof**), V uses at most $r(n)$ random coins and makes at most $q(n)$ non-adaptive queries to locations of π. Then, it accepts or rejects. Let $V^\pi(x)$ denote the random variable representing V's output on input x and with random access to π.
- *Completeness*: If $x \in L$, then $\exists \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] = 1$
- *Soundness*: If $x \notin L$, then $\forall \pi \in \{0,1\}^* : \mathbf{Pr}\left[V^\pi(x) = 1\right] \leq \frac{1}{2}$

We say that a language L is in $\mathbf{PCP}[r(n), q(n)]$ if L has a $(\mathcal{O}(r(n)), \mathcal{O}(q(n)))$-**PCP** verifier.

Main Results

- Obviously:

$$\mathbf{PCP}[0,0] = ?$$
$$\mathbf{PCP}[0,poly] = ?$$
$$\mathbf{PCP}[poly,0] = ?$$

Main Results

- Obviously:

$$\mathbf{PCP}[0,0] = \mathbf{P}$$
$$\mathbf{PCP}[0, poly] = ?$$
$$\mathbf{PCP}[poly, 0] = ?$$

Main Results

- Obviously:

$$\begin{aligned}\mathbf{PCP}[0,0] &= \mathbf{P}\\ \mathbf{PCP}[0,poly] &= \mathbf{NP}\\ \mathbf{PCP}[poly,0] &= ?\end{aligned}$$

Main Results

- Obviously:

$$\begin{aligned}\mathbf{PCP}[0,0] &= \mathbf{P}\\ \mathbf{PCP}[0,poly] &= \mathbf{NP}\\ \mathbf{PCP}[poly,0] &= co\mathbf{RP}\end{aligned}$$

Main Results

- Obviously:

$$\begin{aligned}\mathbf{PCP}[0,0] &= \mathbf{P}\\ \mathbf{PCP}[0,poly] &= \mathbf{NP}\\ \mathbf{PCP}[poly,0] &= co\mathbf{RP}\end{aligned}$$

- A surprising result from Arora, Lund, Motwani, Safra, Sudan, Szegedy states that:

Theorem

$$\mathbf{NP} = \mathbf{PCP}[\log n, 1]$$

Properties

- The restriction that the proof length is at most $q2^r$ is inconsequential, since such a verifier can look on at most this number of locations.
- We have that $\mathbf{PCP}[r(n), q(n)] \subseteq \mathbf{NTIME}[2^{\mathcal{O}(r(n))}q(n)]$, since a NTM could guess the proof in $2^{\mathcal{O}(r(n))}q(n)$ time, and verify it deterministically by running the verifier for all $2^{\mathcal{O}(r(n))}$ possible choices of its random coin tosses. If the verifier accepts for all these possible tosses, then the NTM accepts.

Contents

Why counting?

- So far, we have seen two versions of problems:
  - Decision Problems (if a solution *exists*)
  - Function Problems (if a solution can be *produced*)
- A very important type of problems in Complexity Theory is also:
  - Counting Problems (*how many* solutions exist)

Example (#SAT)

Given a Boolean Expression, compute the number of different truth assignments that satisfy it.

- Note that if we can solve #SAT in polynomial time, we can solve SAT also.
- Similarly, we can define #HAMILTON PATH, #CLIQUE, etc.

Basic Definitions

Definition (**#P**)

A function $f : \{0,1\}^* \to \mathbb{N}$ is in **#P** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing Machine M such that for every $x \in \{0,1\}^*$:

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

- The definition implies that $f(x)$ can be expressed in $poly(|x|)$ bits.
- Each function f in **#P** is equal to the number of paths from an initial configuration to an accepting configuration, or **accepting paths** in the configuration graph of a poly-time NDTM.
- $\mathbf{FP} \subseteq \mathbf{\#P} \subseteq \mathbf{PSPACE}$
- If $\mathbf{\#P} = \mathbf{FP}$, then $\mathbf{P} = \mathbf{NP}$.
- If $\mathbf{P} = \mathbf{PSPACE}$, then $\mathbf{\#P} = \mathbf{FP}$.

Counting Problems

- In order to formalize a notion of completeness for **#P**, we must define proper reductions:

Definition (Cook Reduction)

A function f is **#P**-complete if it is in **#P** and every $g \in$ **#P** is in $\mathbf{FP}^f$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in$ **NP**, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : \; R_A(x,y) = 1\}| \in \#\mathbf{P}$

Counting Problems

- In order to formalize a notion of completeness for **#P**, we must define proper reductions:

Definition (Cook Reduction)

A function f is **#P**-complete if it is in **#P** and every $g \in$ **#P** is in $\mathbf{FP}^f$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in$ **NP**, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : R_A(x,y) = 1\}| \in \#\mathbf{P}$
- We now define a more strict form of reduction:

Counting Problems

Definition (Parsimonious Reduction)

We say that there is a parsimonious reduction from $\#A$ to $\#B$ if there is a polynomial time transformation f such that for all x:

$$|\{y : \ R_A(x,y) = 1\}| = |\{z : \ R_B(f(x),z) = 1\}|$$

- Or, using function notation:

Definition

$$f \leq_m^p g \iff \exists h \in \mathbf{FP} : \ \forall x \, f(x) = g(h(x))$$

Completeness Results

Theorem

#CIRCUIT SAT *is* #**P**-*complete.*

Proof:

- Let $f \in \#\mathbf{P}$. Then, $\exists M, p$:
  $f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$.
- Given x, we want to construct a circuit C such that:

$$|\{z : \ C(z)\}| = |\{y : \ y \in \{0,1\}^{p(|x|}, M(x,y) = 1\}|$$

- We can construct a circuit $\hat{C}$ such that on input x, y simulates $M(x,y)$.
- We know that this can be done with a circuit with size about the square of M's running time.
- Let $C(y) = \hat{C}(x,y)$. □

Completeness Results

Theorem
#SAT *is* #**P**-*complete.*

Proof:

- We reduce #CIRCUIT SAT to #SAT:
- Let a circuit C, with $x_1, \ldots, x_n$ input gates and $1, \ldots, m$ gates.
- We construct a Boolean formula ϕ with variables $x_1, \ldots, x_n, g_1, \ldots, g_m$, where g_i represents the output of gate i.
- A gate can be complete described by simulating the output for each of the 4 possible inputs.
- In this way, we have reduced C to a formula ϕ with $n + m$ variables and $4m$ clauses. $\square$

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_j\} \in E$ iff $A_{i,j} = 1$.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_j\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$ and $\{x_i, y_j\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.
- So, in this case $perm(A)$ is the number of perfect matchings in the corresponding graph!

Valiant’s Theorem

Theorem (Valiant’s Theorem)

*PERMANENT is #***P***-complete under Cook reductions.*

The Class $\oplus\mathbf{P}$

Definition

A language L is in the class $\oplus\mathbf{P}$ if there is a NDTM M such that for all strings x, $x \in L$ iff the *number of accepting paths* on input x is odd.

- The problems $\oplus$SAT and $\oplus$HAMILTON PATH are $\oplus\mathbf{P}$-complete.
- $\oplus\mathbf{P}$ is closed under complement.
- $\oplus\mathbf{P}^{\oplus\mathbf{P}} = \oplus\mathbf{P}$

Operators on Complexity Classes

- So far, we 've defined a lot of *operators* on complexity classes. We will remind them, and define some new in the same way:

Definition (Non-Deterministic Operator)

Let $\mathbf{C}$ be a complexity class. A language $L \in \mathcal{N} \cdot \mathbf{C}$ if there exists $A \in \mathbf{C}$ such that:

- $x \in L \Rightarrow \exists y : x; y \in A$
- $x \notin L \Rightarrow \forall y : x; y \notin A$

- If $\mathbf{C}$ can be expressed using quantifier notation, then the $\mathcal{N}\cdot$ operator adds a $(\exists \cdot /\forall \cdot)$ in front of it.

Example

$\mathcal{N} \cdot \mathbf{P} = \mathbf{NP}$
$\mathcal{N} \cdot \Pi_{i-1}^{p} = \Sigma_{i}^{p}$
$\mathcal{N} \cdot \mathbf{BPP} = \mathbf{MA}$

Operators on Complexity Classes

Definition (Two-sided Probabilistic Operator)

Let $\mathbf{C}$ be a complexity class. A language $L \in \mathcal{BP} \cdot \mathbf{C}$ if there exists $A \in \mathbf{C}$ such that:

- $x \in L \Rightarrow \exists^+ y : x; y \in A$
- $x \notin L \Rightarrow \exists^+ y : x; y \notin A$

Example

$\mathcal{BP} \cdot \mathbf{P} = \mathbf{BPP}$, $\mathcal{BP} \cdot \mathbf{NP} = \mathbf{AM}$

Definition (One-sided Probabilistic Operator)

Let $\mathbf{C}$ be a complexity class. A language $L \in \mathcal{R} \cdot \mathbf{C}$ if there exists $A \in \mathbf{C}$ such that:

- $x \in L \Rightarrow \exists^+ y : x; y \in A$
- $x \notin L \Rightarrow \forall y : x; y \notin A$

Operators on Complexity Classes

Definition

Let $\mathbf{C}$ be a complexity class. A language $L \in \oplus \cdot \mathbf{C}$ if there exists $A \in \mathbf{C}$ such that:

$$x \in L \Leftrightarrow |\{y : x; y \in A\}| \text{ is odd}$$

Example

$\oplus \cdot \mathbf{P} = \oplus\mathbf{P}$

Remark

Note that the class $\mathbf{C}$ in the above definitions must be closed under padding.

Valiant-Vazirani Theorem

Theorem (Valiant-Vazirani)

Given a Boolean Formula ϕ in CNF, it can be transformed by a probabilistic, polynomial-time algorithm to a formula ϕ', such that:

- $\phi \in \text{SAT} \Longrightarrow \Pr\left[\phi' \in \oplus\text{SAT}\right] > \frac{1}{p(|\phi|)}$
- $\phi \notin \text{SAT} \Longrightarrow \phi' \notin \oplus\text{SAT}$

The above is equivalent with:

Theorem (Valiant-Vazirani)

$$\mathbf{NP} \subseteq \mathcal{R} \cdot \oplus\mathbf{P}$$

- It also implies that $\mathbf{NP} \subseteq \mathbf{RP}^{\text{USAT}}$, where USAT is the unique-satisfiability problem.

Proof of Valiant-Vazirani Theorem

Proof:

- Let $\phi = \phi(x_1, \ldots, x_n)$.
- Let S be a random subset of $[n] = \{1, \ldots, n\}$.
  (*uses n random bits*).
- Let $[S] = \oplus_{i \in S} x_i$.
- The reduction algorithm is the following:
  - Input ϕ.
  - Guess Randomly $k \in \{0, \ldots, n-1\}$.
  - Guess Randomly subsets $S_1, \ldots, S_{k+2} \subseteq [n]$.
  - Output $\phi' = \phi \wedge [S_1] \wedge [S_2] \wedge \cdots \wedge [S_{k+2}]$.

Proof of Valiant-Vazirani Theorem

Proof:

- Let $\phi = \phi(x_1, \ldots, x_n)$.
- Let S be a random subset of $[n] = \{1, \ldots, n\}$.
  (*uses n random bits*).
- Let $[S] = \oplus_{i \in S} x_i$.
- The reduction algorithm is the following:
  - Input ϕ.
  - Guess Randomly $k \in \{0, \ldots, n-1\}$.
  - Guess Randomly subsets $S_1, \ldots, S_{k+2} \subseteq [n]$.
  - Output $\phi' = \phi \wedge [S_1] \wedge [S_2] \wedge \cdots \wedge [S_{k+2}]$.
- With each addition of a subformula of the form $[S_i]$ to the conjunction, the number of satisfying assignments is halved, since for each assignment b the probability that $b([S]) = 0$ is $1/2$.

Proof of Valiant-Vazirani Theorem

Proof (*cont'd*):

- These events are only pairwise independent.
- If ϕ is **unsatisfiable**, then ϕ' is clearly unsatisfiable, therefore $\phi' \notin \oplus$SAT.
- If ϕ is **satisfiable**, let $m \geq 1$ the number of satisfying assignments.

Proof of Valiant-Vazirani Theorem

Proof (*cont'd*):

- These events are only pairwise independent.
- If ϕ is **unsatisfiable**, then ϕ' is clearly unsatisfiable, therefore $\phi' \notin \oplus$SAT.
- If ϕ is **satisfiable**, let $m \geq 1$ the number of satisfying assignments.
- With probability $\geq 1/n$, k will be chosen so that: $2^k \leq m \leq 2^{k+1}$.
- For that fixed k, let b be a fixed satisfying assignment of ϕ.
- Since $[S_i]$'s are chosen *independently*,

$$\Pr\left[b(\phi') = 1\right] = \frac{1}{2^{k+2}}$$

Proof of Valiant-Vazirani Theorem

Proof (*cont'd*):

- Even if b ”survived“ the conjunction process, the probability that any other satisfying assignment b' of ϕ also survives the conjuction is also $1/2^{k+2}$.
- The probability that b is the **only** formula that survives the conjuction (*cf.* USAT):

$$\frac{1}{2^{k+2}} \cdot \left(1 - \sum_{b'} \frac{1}{2^{k+2}}\right) = \frac{1}{2^{k+2}} \cdot \left(1 - \frac{m-1}{2^{k+2}}\right) \geq$$

$$\geq \frac{1}{2^{k+2}} \cdot \left(1 - \frac{2^{k+1}}{2^{k+2}}\right) = \frac{1}{2^{k+3}}$$

Proof of Valiant-Vazirani Theorem

Proof (*cont'd*):

- Thus, the probability that *there is a b* that is the **only** satisfying assignment of ϕ' is *at least*:

$$\sum_b \frac{1}{2^{k+3}} = \frac{m}{2^{k+3}} \geq \frac{2^k}{2^{k+3}} = \frac{1}{8}$$

- So, we proved that for this choice of k, the probability is at least $1/8$.
- Thus,

$$\Pr\left[\phi' \notin \oplus\mathrm{SAT}\right] \geq \frac{1}{n} \cdot \frac{1}{8} = \frac{1}{8n}$$

$\square$

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and **#P**.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and **#P**.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Theorem (Toda's Theorem)

$$\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{\#P}[1]}$$

Proof of Toda's Theorem

Proof Sketch

The proof consists of two main lemmas:

- **Lemma 1**: $\boxed{\mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus\mathbf{P}}$
- **Lemma 2**: $\boxed{\mathcal{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}}$

Proof of Toda's Theorem

Proof Sketch

The proof consists of two main lemmas:

- **Lemma 1**: $\boxed{\mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus\mathbf{P}}$
  - We have to prove that for a complexity class **C**:

$$\oplus \cdot \oplus \cdot \mathbf{C} = \oplus \cdot \mathbf{C}$$

$$\mathcal{BP} \cdot \mathcal{BP} \cdot \mathbf{C} = \mathcal{BP} \cdot \mathbf{C}$$

  - And then prove that under certain conditions they can be commuted:

$$\oplus \cdot \mathcal{BP} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$

- **Lemma 2**: $\boxed{\mathcal{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}}$

Proof of Toda's Theorem

Proof Sketch

The proof consists of two main lemmas:

- **Lemma 1**: $\boxed{\mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus\mathbf{P}}$
  - We have to prove that for a complexity class $\mathbf{C}$:
$$\oplus \cdot \oplus \cdot \mathbf{C} = \oplus \cdot \mathbf{C}$$
$$\mathcal{BP} \cdot \mathcal{BP} \cdot \mathbf{C} = \mathcal{BP} \cdot \mathbf{C}$$
  - And then prove that under certain conditions they can be commuted:
$$\oplus \cdot \mathcal{BP} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$
- **Lemma 2**: $\boxed{\mathcal{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}}$
  - We have to ”transform“ somehow a probabilistic parity machine, so we can decide every language by counting paths, by a single **#P** oracle call.

Proof of Toda's Theorem

Lemma 1.1

$$\oplus \cdot \oplus \cdot \mathbf{C} = \oplus \cdot \mathbf{C}$$

Proof

- Let $L \in \mathbf{C}$, $L' \in \oplus \cdot \mathbf{C}$ and $L'' \in \oplus \cdot \oplus \cdot \mathbf{C}$.
- $x \in '' \Leftrightarrow |\{y_1 : x; y_1 \in L'\}|$ is odd $\Leftrightarrow \sum_{y_1} L'(x; y_1) \equiv 1 \mod 2$

  $\Leftrightarrow \sum_{y_1} \sum_{y_2} L(x; y_1; y_2) \equiv 1 \mod 2$

  $\Leftrightarrow \sum_{y_1, y_2} L(x; y_1; y_2) \equiv 1 \mod 2$

  $\Leftrightarrow |\{y_1; y_2 : x; y_1; y_2 \in L\}|$ is odd $\Leftrightarrow x \in L'$ $\square$

Proof of Toda's Theorem

Lemma 1.2

$$\mathcal{BP} \cdot \mathcal{BP} \cdot \mathbf{C} = \mathcal{BP} \cdot \mathbf{C}$$

Proof:
Easy exercise :)

Proof of Toda's Theorem

Lemma 1.2

$$\mathcal{BP} \cdot \mathcal{BP} \cdot \mathbf{C} = \mathcal{BP} \cdot \mathbf{C}$$

Proof:
Easy exercise :)

Lemma 1.3

$$\oplus \cdot \mathcal{BP} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$

Proof of Toda's Theorem

Lemma 1.3

$$\oplus \cdot \mathcal{BP} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$

Proof:

- Let $L \in \oplus \cdot \mathcal{BP} \cdot \mathbf{C}$.
- Then $\exists A \in \mathcal{BP} \cdot \mathbf{C}$, such that:

$$x \in L \Leftrightarrow \left|\{z : \ |z| = |x|^k \ \wedge \ x; z \in A\}\right| \text{ is odd}$$

- Then, $\exists B \in \mathbf{C}$, such that:

$$\Pr_w\left[\exists z \in \{0,1\}^{|x|^k} : \ x; z; w \in B \Leftrightarrow x; z \notin A\right] \leq \frac{1}{3}$$

Proof of Toda's Theorem

Proof (*cont'd*):

- Let $B' = \{x;w;z : \ x;z;w \in B\} \in \mathbf{C}$.
- Let $B'' = \{x;w : \ |\{z : \ |z| = |x|^k \ \wedge \ x;w;z \in B'\}| \text{ is odd}\} \in \oplus \cdot \mathbf{C}$.
- $\boxed{x \in L} \Rightarrow |\{z : \ |z| = |x|^k \ \wedge \ x;z \in A\}| \text{ is odd}$
  $\Rightarrow \Pr_w \left[|\{z : \ |z| = |x|^k \ \wedge \ x;z;w \in B\}| \text{ is odd}\right] \geq \frac{2}{3}$
  $\Rightarrow \Pr_w \left[x;w \in B''\right] \geq \frac{2}{3}$
- $\boxed{x \notin L} \Rightarrow |\{z : \ |z| = |x|^k \ \wedge \ x;z \in A\}| \text{ is even}$
  $\Rightarrow \Pr_w \left[|\{z : \ |z| = |x|^k \ \wedge \ x;z;w \in B\}| \text{ is odd}\right] \leq \frac{1}{3}$
  $\Rightarrow \Pr_w \left[x;w \in B''\right] \leq \frac{1}{3}$
- Hence, $L \in \mathbf{BP} \cdot \oplus \cdot \mathbf{C}$. □

Proof of Toda's Theorem

Lemma 1.4

$$\mathcal{N} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$

Proof Idea:

- That is, essentially, a generalization of Valiant-Vazirani Theorem:
- Instead of SAT, we could use Σ_k^p-complete version of SAT_k and prove with slight modifications that:

$$\Sigma_k^p = \mathcal{N} \cdot \Pi_{k-1}^p \subseteq \mathcal{BP} \cdot \oplus \cdot \Pi_{k-1}^p$$

Proof of Toda's Theorem

Lemma 1

$$\mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus\mathbf{P}$$

Proof (*of Lemma 1*):

- We will prove by induction that $\Sigma_k^p, \Pi_k^p \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$
- The base $k = 0$ is trivial, since $\mathbf{P} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$.
- The induction hypothesis states that $\Sigma_{k-1}^p, \Pi_{k-1}^p \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$.
- Then:

$$\Sigma_k^p = \mathcal{N} \cdot \Pi_{k-1} \subseteq \mathcal{BP} \cdot \oplus \cdot \Pi_{k-1}^p \subseteq \mathcal{BP} \cdot \oplus \cdot \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$$

$$\subseteq \mathcal{BP} \cdot \mathcal{BP} \cdot \oplus \cdot \oplus \cdot \mathbf{P} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$$

□

Proof of Toda's Theorem

Lemma 2

$$\mathcal{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$$

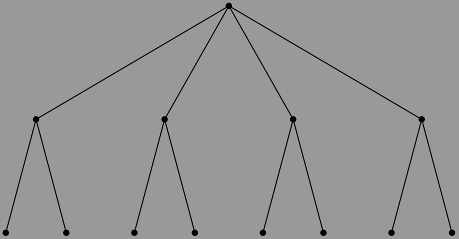
Proof Sketch:

- Let $L \in \mathcal{BP} \cdot \oplus\mathbf{P}$
- So, $\exists A \in \oplus\mathbf{P}$, such that:

$$\mathbf{Pr}_y[x \in L \Leftrightarrow x; y \in A] \geq \frac{2}{3}$$

Proof of Toda's Theorem

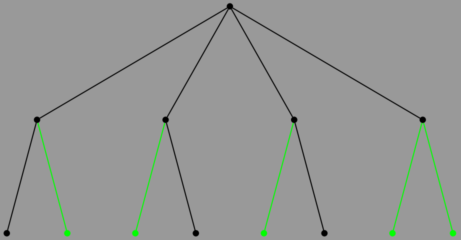
Amplification Example



- Example $\mod 8$.
- We want to modify this tree to another s.t.:
  - Odd number of z's $\Longrightarrow$ number of z''s $\equiv 0 \mod 8$
  - Even number of z's $\Longrightarrow$ number of z''s $\equiv 1 \mod 8$

Proof of Toda's Theorem

Amplification Example

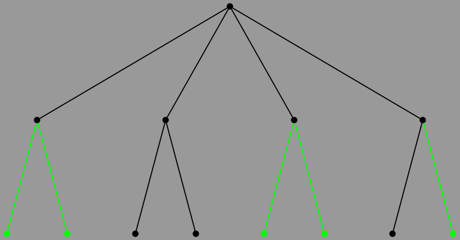


$\boxed{x \in L}$

- Example $\mod 8$.
- We want to modify this tree to another s.t.:
  - Odd number of z's $\Longrightarrow$ number of z''s $\equiv 0 \mod 8$
  - Even number of z's $\Longrightarrow$ number of z''s $\equiv 1 \mod 8$

Proof of Toda's Theorem

Amplification Example



$\boxed{x \notin L}$

- Example $\mod 8$.
- We want to modify this tree to another s.t.:
    - Odd number of z's $\Longrightarrow$ number of z''s $\equiv 0 \mod 8$
    - Even number of z's $\Longrightarrow$ number of z''s $\equiv 1 \mod 8$

Proof of Toda's Theorem

- Now, let $g = g(x)$ be the **number of accepting computations**:
  - If $g \equiv 0 \mod 8$ or $g \equiv 1 \mod 8$, then $x \in A$.
  - If $g \equiv 3 \mod 8$ or $g \equiv 4 \mod 8$, then $x \notin A$.
- We can generalize this so that:

$$x \in A \Leftrightarrow g < \frac{2^{p(|x|)}}{2} \mod 2^{p(|x|)}$$

Lemma 2.1

For $A \in \oplus\mathbf{P}$, and $\forall p \in poly(n)$, $\exists$ PNTM M:

- $x \in A \Rightarrow \#acc_M(x) \equiv 0 \mod 2^{p(n)}$
- $x \notin A \Rightarrow \#acc_M(x) \equiv 1 \mod 2^{p(n)}$

Proof of Toda's Theorem

- Let:

$$h(x) = \sum_{y,|y|=p(|x|)} \#acc_M(x;y)$$

$$= \sum_{x;y\in A} \#acc_M(x;y) + \sum_{x;y\notin A} \#acc_M(x;y)$$

$$\equiv -g(x) \mod 2^{p(n)}$$

- So, we can decide $x \in L$ from $h(x)$.
- But, $h \in$ **#P**: *on input* x*, guess a* y*,* $|y| = p(|x|)$*, and simulate* M *on* $x; y$.
- Hence $L \in \mathbf{P}^{\#\mathbf{P}[1]}$.

The Class GapP

- For a TM M, we define:

$$\Delta M(x) = \#acc(x) - \#rej(x) = \#M(x) - \#\overline{M}(x)$$

Definition

A function $f : \{0,1\}^* \to \mathbb{N}$ is in **GapP** if there exists a poly-time NDTM M such that for all inputs x:

$$f(x) = \Delta M(x)$$

- **GapP** functions are **closed under negation**:
  $f \in \textbf{GapP} \Rightarrow -f \in \textbf{GapP}$.
- **GapP**, unlike **#P**, encompasses <u>all</u> **FP** functions.

The Class GapP

Theorem

For all functions f, the following are equivalent:

1. $f \in \mathbf{GapP}$.
2. *f is the difference of two $\#\mathbf{P}$ functions.*
3. *f is the difference of a $\#\mathbf{P}$ and an $\mathbf{FP}$ function.*
4. *f is the difference of a $\mathbf{FP}$ and an $\#\mathbf{P}$ function.*

In other words:

$$\mathbf{GapP} = \#\mathbf{P} - \#\mathbf{P} = \#\mathbf{P} - \mathbf{FP} = \mathbf{FP} - \#\mathbf{P}$$

- $(3) \Rightarrow \mathbf{GapP} \subseteq \mathbf{FP}^{\#\mathbf{P}[1]}$.

Characterizations of Complexity Classes

- **NP** consists of those languages L such that for some **#P** function f and all inputs x:
  - If $x \in L$ then $f(x) > 0$.
  - If $x \notin L$ then $f(x) = 0$.
- **UP** consists of those languages L such that for some **#P** function f and all inputs x:
  - If $x \in L$ then $f(x) = 1$.
  - If $x \notin L$ then $f(x) = 0$.
- **PP** consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) > 0$.
  - If $x \notin L$ then $f(x) \leq 0$ (of $f(x) < 0$).
- **SPP** consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) = 1$.
  - If $x \notin L$ then $f(x) = 0$.

Characterizations of Complexity Classes

- $\mathbf{C_=P}$ consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) = 0$.
  - If $x \notin L$ then $f(x) \neq 0$ (or $f(x) > 0$).
- $\oplus$**P** consists of those languages L such that for some **#P** function f and all inputs x:
  - If $x \in L$ then $f(x)$ is odd.
  - If $x \notin L$ then $f(x)$ is even.
- $\mathbf{Mod_kP}$ consists of those languages L such that for some **#P** function f and all inputs x:
  - If $x \in L$ then $f(x) \mod k \neq 0$.
  - If $x \notin L$ then $f(x) \mod k = 0$.
- **MiddleP** consists of those languages L such that for some **#P** function f and all inputs x:
  - If $x \in L$ then $\mathit{middle}(f(x)) = 1$.
  - If $x \notin L$ then $\mathit{middle}(f(x)) = 0$.

Characterizations of Complexity Classes

- We can summarize the above:

Class	Function f in:	If $x \in L$:	If $x \notin L$:
NP	**#P**	$f(x) > 0$	$f(x) = 0$
UP	**#P**	$f(x) = 1$	$f(x) = 0$
PP	**GapP**	$f(x) > 0$	$f(x) \leq 0$ or $f(x) < 0$
SPP	**GapP**	$f(x) = 1$	$f(x) = 0$
$\mathbf{C_{=}P}$	**GapP**	$f(x) = 0$	$f(x) \neq 0$ or $f(x) > 0$
$\oplus$**P**	**#P**	$f(x)$ is odd	$f(x)$ is even
$\mathbf{Mod_k P}$	**#P**	$f(x) \mod k \neq 0$	$f(x) \mod k = 0$
MiddleP	**#P**	$middle(f(x)) = 1$	$middle(f(x)) = 0$

Characterizations of Complexity Classes

- We define *middle* : $\{0,1\}^* \to \{0,1\}$ to return the $\lceil \frac{|x|}{2} \rceil^{th}$ bit of the string x.
- The class **MiddleP** considers the **middle bit** of a string, as **PP** consider the **high-order bit** and $\oplus$**P** the **low-order bit**.
- Observe that $\oplus\mathbf{P} = \mathbf{Mod_2P}$.
- From the above we can easily have:
  - $\mathbf{NP} \subseteq co\mathbf{C_{=}P} \subseteq \mathbf{PP}$
  - $\mathbf{UP} \subseteq \mathbf{SPP}$
  - $\mathbf{C_{=}P} \subseteq \mathbf{PP}$
  - **PP** is closed under complement.

Characterizations of Complexity Classes

Theorem

$$\mathbf{P}^{\mathbf{PP}} = \mathbf{P}^{\mathbf{GapP}}$$

Proof:

- We only need to show that every **GapP** function g is computable in $\mathbf{FP}^{\mathbf{PP}}$.
- Consider the **GapP** function $f(x,k) = g(x) - k$.
- Then $L = \{\langle x,k\rangle : \ g(x) > k\} \in \mathbf{PP}$, by the previous classification.
- Use *binary search* using L as an oracle to find the value of $g(x)$. $\square$

Summary

- **#P** is the class of *functions* counting accepting paths of NPTMs.
- Any **NP** problem (hence any **P** problem) has an associated counting problem.
- #SAT is #**P**-complete under parsimonious reductions.
- PERMANENT is #**P**-complete under Cook reductions.
- Valiant-Vazirani theorem states that we can reduce SAT to USAT (Unique Satisfiability) with a one-sided error randomized reduction.
- Toda's theorem states that we can decide *any* language in **PH** by a *single* oracle call to a #**P** function. So, counting is harder than quantifying.