Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)
Αλγόριθμοι & Πολυπλοκότητα ΙΙ (ΜΠΛΑ)

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών
Εθνικό Μετσόβιο Πολυτεχνείο

2015-2016

Πληροφορίες Μαθήματος

Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ)
Αλγόριθμοι & Πολυπλοκότητα ΙΙ (ΜΠλ∀), Λ4-Υπ.

- Διδάσκοντες: Σ. Ζάχος, Ά. Παγουρτζής
- Βοηθοί Διδασκαλίας: Α. Αντωνόπουλος, Α. Χαλκή
- Επιμέλεια Διαφανειών: Α. Αντωνόπουλος
- Δευτέρα: 17:00 - 19:00 (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, Παρασκευή 11:00-13:00
- Σελίδα: `www.corelab.ntua.gr/courses/complexity/`
- Βαθμολόγηση:

Διαγώνισμα:	6 μονάδες
Ασκήσεις:	2 μονάδες
Ομιλία:	2 μονάδες
Quizes :	2 μονάδες

Computational Complexity

Graduate Course

Computation and Reasoning Laboratory
National Technical University of Athens

2015-2016

Professors:
S. Zachos, Professor
A. Pagourtzis, Ass. Professor

Slides: Antonis Antonopoulos

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. E. Allender, M. Loui, and K. Regan, **Three chapters for the CRC Handbook on Algorithms and Theory of Computation** (M.J. Atallah, ed.), (Boca Raton: CRC Press, 1998).

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

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.
- 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*).

Efficiently Computable $\equiv$ Polynomial-Time Computable

Contents

Definition

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

- $Q = \{q_0, q_1, q_2, q_3, \ldots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ 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.
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{S, L, R\}$ 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 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 $T : \mathbb{N} \to \mathbb{N}$. 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 $S : \mathbb{N} \to \mathbb{N}$. 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].

- 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].

- 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.

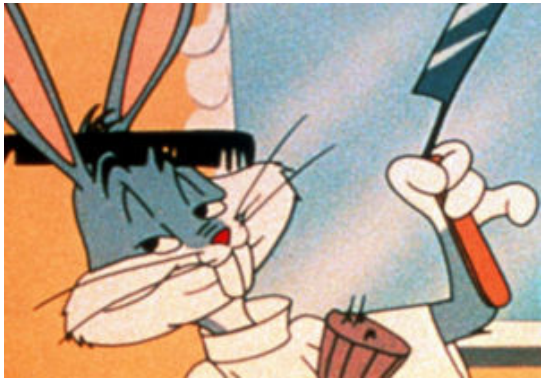
Definition

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**.

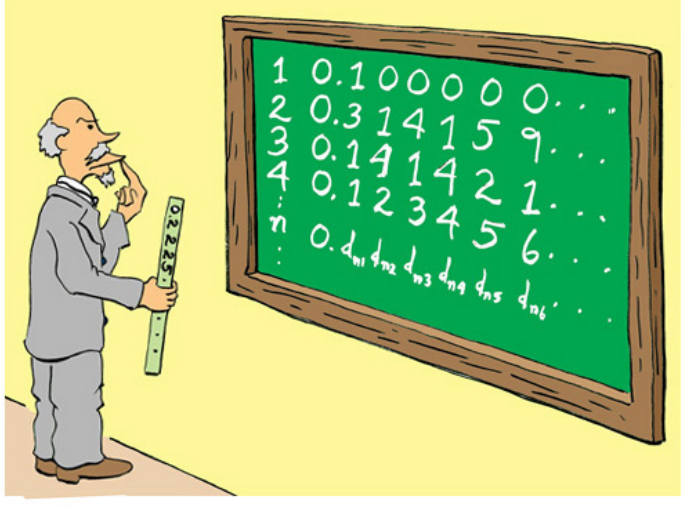
Contents

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 $\phi_y(y) = \phi_y(y) + 1$. $\square$

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 Turing Machine.**
- **Every TM is represented by infinitely many strings.**

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 indepedent 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: $\mathtt{H} = \{\llcorner M \lrcorner; x \mid M(x) \downarrow\}$, 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

$\mathtt{H}$ *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 $\mathtt{H}$.
- Consider the TM D:

$$D(\llcorner M \lrcorner) : \textbf{ if } M_H(\llcorner M \lrcorner; \llcorner M \lrcorner) = \text{“yes”} \textbf{ then } \uparrow \textbf{ else } \text{“yes”}$$

- What is $D(\llcorner D \lrcorner)$?
- If $D(\llcorner D \lrcorner) \uparrow$, then M_H accepts the input, so $\llcorner D \lrcorner; \llcorner D \lrcorner \in \mathtt{H}$, so $D(D) \downarrow$.
- If $D(\llcorner D \lrcorner) \downarrow$, then M_H rejects $\llcorner D \lrcorner; \llcorner D \lrcorner$, so $\llcorner D \lrcorner; \llcorner D \lrcorner \notin \mathtt{H}$, 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

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

Rice's Theorem

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

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 deciding the HALTING PROBLEM for an arbitrary input x. For each $x \in \Sigma^*$, we construct a TM M as follows:

  $M(y)$: **if** $M_H(x) =$ "yes" **then** $M_L(y)$ **else** $\uparrow$

- We claim that: $L(M) \in \mathcal{C}$ if and only if $x \in \mathtt{H}$.
  Proof of the claim:
  - If $x \in \mathtt{H}$, then $M_H(x) =$ "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 $\mathcal{C}$. $\square$

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 $\mathcal{O}(T(n))$, 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 $\mathcal{O}(S(n))$, 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 $\mathcal{O}(T(n))$.
- We say that $L \in \mathbf{NSPACE}[S(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within space bound $\mathcal{O}(S(n))$.

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 (Complement of a 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?

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.

- 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.
- 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 $f_1^{f_2}$.

Complexity Classes

Constructible Functions

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

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

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 U(M_x, x);
If 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, $D(x_M) = 1 - M(x_M)$ $\mathbf{D(x_M) = 1 - M(x_M)}$ (while we have also that $D(x) = M(x), \ \forall x$)
Contradiction!! □

- 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:

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. $\mathbf{NSPACE}[S(n)] \subseteq \mathbf{DTIME}[k^{\log n + S(n)}]$

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 $nc_1 = c_1^{\log n + S(n)}$

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.

- 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 $c_2 c_1^{2(\log n + S(n))} \sim k^{\log n + S(n)}$ steps. $\square$

*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

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

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

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

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

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

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 $k \geq 1$.

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**.

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 $L \in \mathbf{NP}$, 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. $\square$

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 date, if one exists.

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

Complements of complexity classes

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

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

- 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:

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

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

Savitch's Theorem

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

Theorem (Savitch's Theorem)

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

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

$PATH(x, y, i)$: "*There is a path from* x *to* y*, of length* $\leq 2^i$" .

- We can solve REACHABILITY if we can compute $PATH(x, y, \lceil \log n \rceil)$, for any nodes $x, y \in V$, since any path in G can be at most $n \log n$ long.
- If $i = 0$, we can check whether $PATH(x, y, i)$.
- If $i \geq 1$:

forall nodes z test whether $PATH(x, z, i-1)$ and $PATH(z, y, i-1)$

Savitch's Theorem

Proof (*cont'd*):

- We generate all nodes z one after the other, *reusing* space.
- Once a z is generated, we add $(x, z, i-1)$ to the tape, and start working on this recursively.
- If a negative answer is obtained to $PATH(x, z, i-1)$, we erase this triple and move to the next z.
- If a positive answer is obtained to $PATH(x, z, i-1)$, we erase the triple and move to $PATH(z, y, i-1)$.
- If this is negative, we erase it and move to the next z.
- If it is positive, we compare it to (x, y, i) to check that this is the second recursive call, and then return a positive answer to $PATH(x, y, i)$.
- The work tape contains at any moment at most $\lceil \log n \rceil$, each of length at most $3 \log n$. $\square$

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

PSPACE $=$ **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.

Definition

A language L_1 is **logspace reducible** to a language L_2, denoted $L_1 \leq_l L_2$, 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 L is **NL**-complete if it is in **NL** and for every $A \in \mathbf{NL}$, $A \leq_l L$.

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)

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

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 $C_1, \ldots, C_n$
- 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 $t \notin C_n$.
- **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 $|C_{i-1}|$.

Certifying that $|C_i| = c$, given $|C_{i-1}|$
The certificate will consist of n certificates for each vertex.
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. $\square$

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,

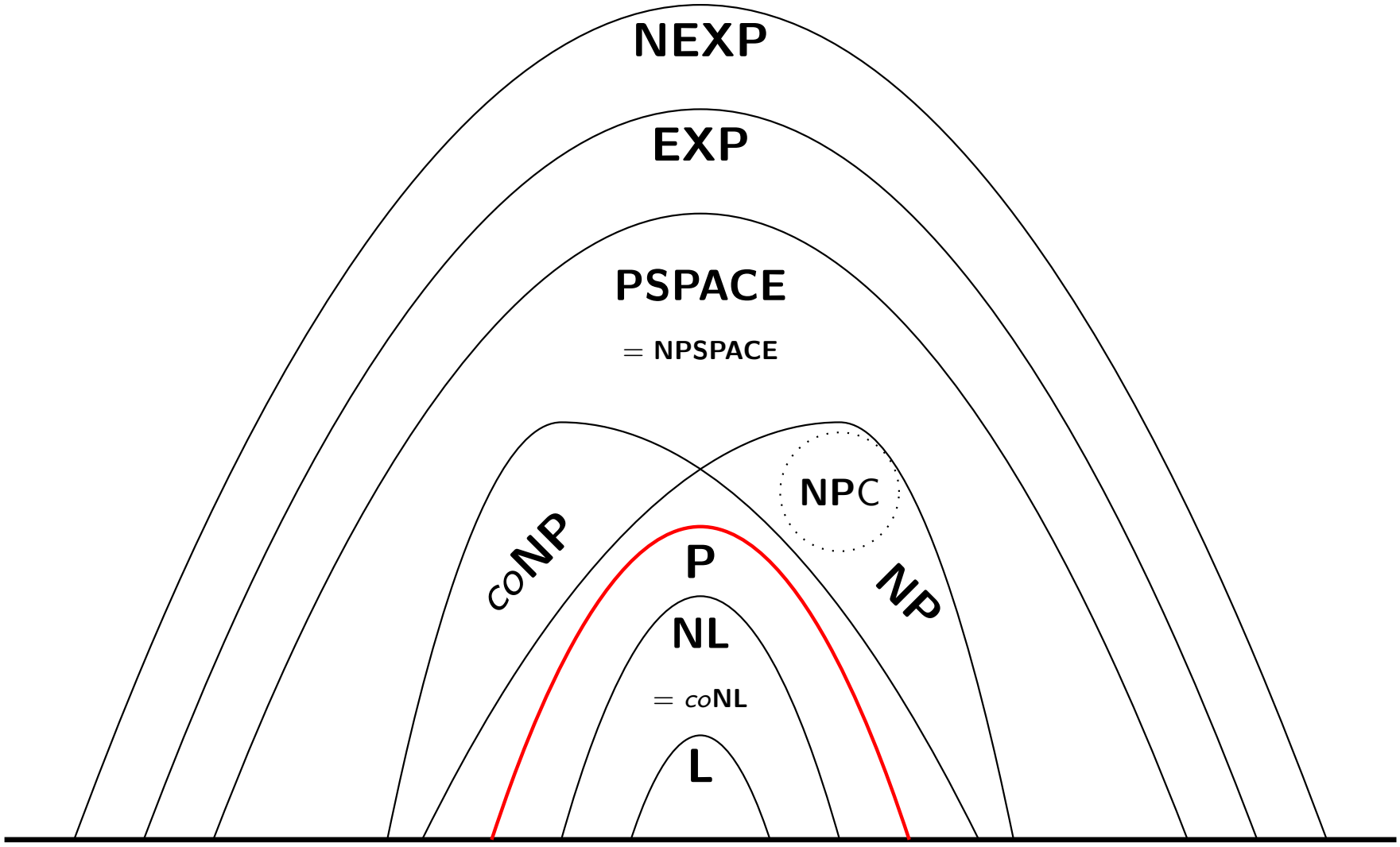
$$\boxed{\mathbf{L} \subseteq \mathbf{SL} \subseteq \mathbf{NL}}$$

- 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:

Theorem (Reigold, 2004)

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

Our Complexity Hierarchy Landscape



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.

- 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 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 $\mathbf{P}^{\mathtt{SAT}} = \mathbf{P}^{\mathbf{NP}}$.

Theorem

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

Proof

Take A to be a **PSPACE**-complete language.Then:
$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{NPSPACE} \subseteq \mathbf{PSPACE}$. $\square$

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 [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 $M_1^?, M_2^?, \ldots$ 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* $x \in B$"?
- **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. $\square$

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 $\mathbf{P}$ vs. $\mathbf{NP}$ is a **nonrelativizing** result, so diagonalization and any other relativizing method doesn't suffice to prove it.

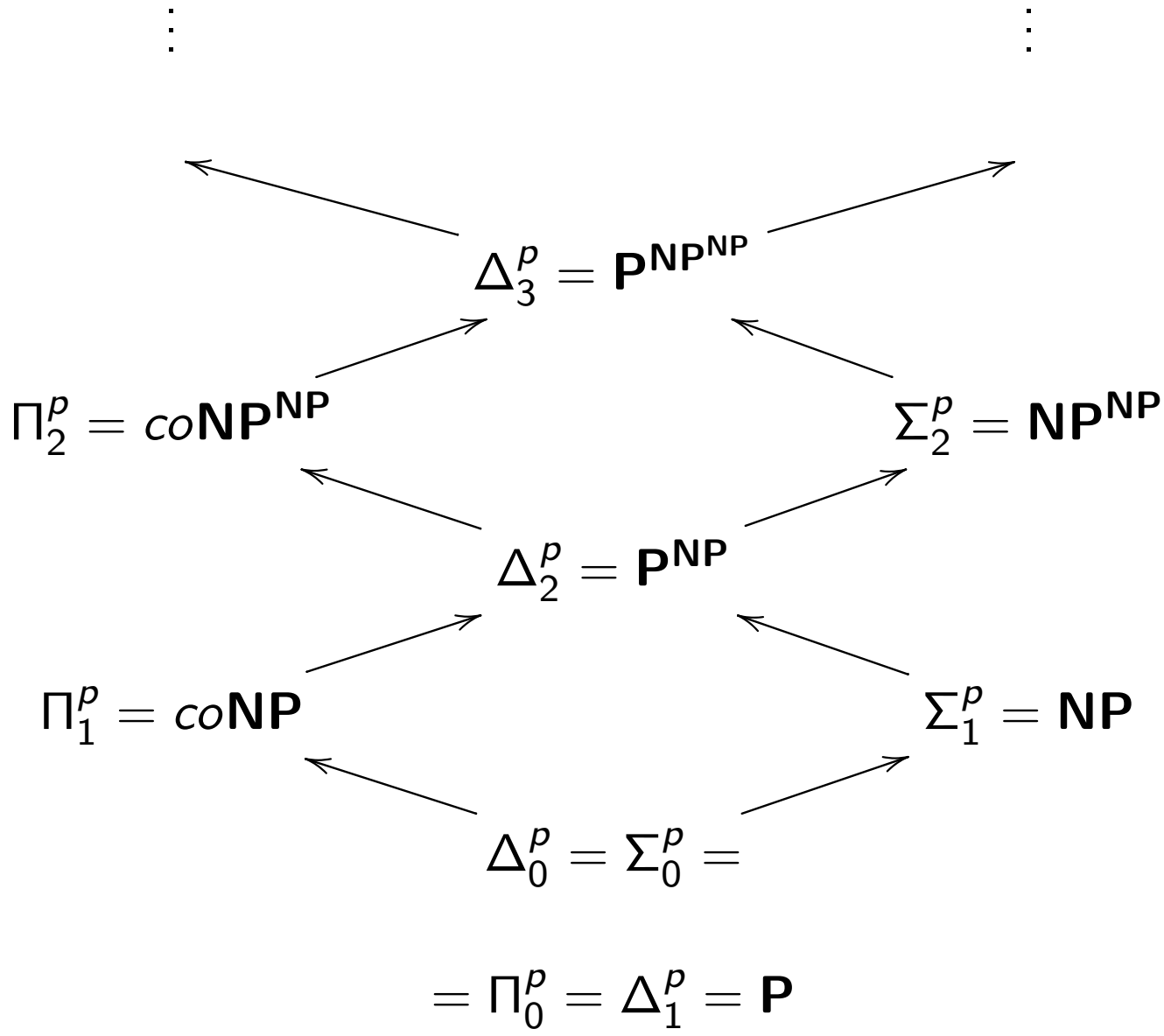
The Polynomial Hierarchy

Polynomial Hierarchy Definition

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

$$\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$$

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


$\Delta_3^p = \mathbf{P}^{\mathbf{NP}^{\mathbf{NP}}}$
$\Pi_2^p = co\mathbf{NP}^{\mathbf{NP}}$
$\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$
$\Delta_2^p = \mathbf{P}^{\mathbf{NP}}$
$\Pi_1^p = co\mathbf{NP}$
$\Sigma_1^p = \mathbf{NP}$
$\Delta_0^p = \Sigma_0^p =$
$= \Pi_0^p = \Delta_1^p = \mathbf{P}$

Basic Theorems

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)

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

Basic Theorems

Proof (cont.)

- 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 comp 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* y *records 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 $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$.

Basic Theorems

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.

Basic Theorems

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

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

Basic Theorems

Corollary

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

$\texttt{QSAT}_i$ Definition

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

$$\exists X_1 \forall X_2 \exists X_3 ..... Q X_i \phi$$

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

Theorem

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

Basic Theorems

Theorem

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

Proof

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)

Contents

Warmup: Randomized Quicksort

Deterministic QuicksortRandomized Quicksort

Input: A list L of integers;

If $n \leq 1$ then return L.

Else {

- let $i = 1$; choose a random integer i, $1 \leq i \leq n$;
- let L_1 be the sublist of L whose elements are $< a_i$;
- let L_2 be the sublist of L whose elements are $= a_i$;
- let L_3 be the sublist of L whose elements are $> a_i$;
- Recursively Quicksort L_1 and L_3;
- return $L = L_1 L_2 L_3$;

Warmup: Randomized Quicksort

- Let T_d the *max* number of comparisons for the Deterministic Quicksort:

$$T_d(n) \geq T_d(n-1) + \mathcal{O}(n)$$

$$\Downarrow$$

$$T_d(n) = \Omega(n^2)$$

- Let T_r the *expected* number of comparisons for the Randomized Quicksort:

$$T_r \geq \frac{1}{n} \sum_{j=0}^{n-1} [T_r(j) - T_r(n-1-j)] + \mathcal{O}(n)$$

$$\Downarrow$$

$$T_r(n) = \mathcal{O}(n \log n)$$

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 $r_1, \ldots, r_n$ from a set $S \subseteq \mathbb{F}$.
5. We are wrong with a probability at most:

Theorem (Schwartz-Zippel Lemma)

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

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

Warmup: Polynomial Identity Testing

Proof:

(*By Induction on n*)

- For $n = 1$: $\mathbf{Pr}[Q(r) = 0 | Q(x) \neq 0] \leq d/|S|$
- For n:

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

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*!
- 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 $\mathcal{O}(T(|x|))$ 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 $x \in \{0,1\}^*$:

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

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

Quantifier Characterizations

- Proper formalism (*Zachos et al.*):

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:

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

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

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***RP**.

Quantifier Characterizations

- $\mathbf{RP} \subseteq \mathbf{NP}$, since every accepting "branch" is a certificate!
- $\mathbf{RP} \subseteq \mathbf{BPP}$, $co\mathbf{RP} \subseteq \mathbf{BPP}$

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

Theorem (Decisive Characterization of BPP)

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

Quantifier Characterizations

Proof:

- Let $L \in \mathbf{BPP}$. Then, by definition, there exists a polynomial-time computable predicate Q and a polynomial q such that for all x's of length n:

$$x \in L \Rightarrow \exists^{+} y \ Q(x, y)$$

$$x \notin L \Rightarrow \exists^{+} y \ \neg Q(x, y)$$

Swapping Lemma

i. $\forall y \exists^{+} z \ R(x, y, z) \Rightarrow \exists^{+} C \forall y \ \bigvee_{z \in C} R(x, y, z)$
ii. $\forall z \exists^{+} y \ R(x, y, z) \Rightarrow \forall C \exists^{+} y \ \bigwedge_{z \in C} R(x, y, z)$

- By the above Lemma: $x \in L \Rightarrow \exists^{+} z \ Q(x, z) \Rightarrow \forall y \exists^{+} z \ Q(x, y \oplus z) \Rightarrow \exists^{+} C \forall y \, [\exists (z \in C) \ Q(x, y \oplus z)]$, where C denotes (as in the Swapping's Lemma formulation) a set of $q(n)$ strings, each of length $q(n)$.

Quantifier Characterizations

Proof (cont'd):

- On the other hand, $x \notin L \Rightarrow \exists^{+} y\ \neg Q(x, z) \Rightarrow \forall z \exists^{+} y\ \neg Q(x, y \oplus z) \Rightarrow \forall C \exists^{+} y\, [\forall (z \in C)\ \neg Q(x, y \oplus z)]$.
- Now, we only have to assure that the appeared predicates $\exists z \in C\ Q(x, y \oplus z)$ and $\forall z \in C\ \neg Q(x, y \oplus z)$ are computable in polynomial time
- Recall that in Swapping Lemma's formulation we demanded $|C| \leq p(n)$ and that for each $v \in C:\ |v| = p(n)$. This means that we seek if a string of polynomial length *exists*, or if the predicate holds *for all* such strings in a set with polynomial cardinality, procedure which can be surely done in polynomial time.

Quantifier Characterizations

Proof (cont'd):

- Conversely, if $L \in (\exists^+\forall/\forall\exists^+)$, for each string w, $|w| = 2p(n)$, we have $w = w_1w_2$, $|w_1| = |w_2| = p(n)$. Then:
  $x \in L \Rightarrow \exists^+ y \forall z\ R(x, y, z) \Rightarrow \exists^+ w\ R(x, w_1, w_2)$
  $x \notin L \Rightarrow \forall y \exists^+ z\ R(x, y, z) \Rightarrow \exists^+ w\ \neg R(x, w_1, w_2)$
- So, $L \in \mathbf{BPP}$. $\square$

- 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)

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

BPP and PH

Theorem (Sipser-Gács)

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

Proof (*Lautemann*)
Because *co*$\mathbf{BPP} = \mathbf{BPP}$,we prove only $\mathbf{BPP} \subseteq \Sigma_2\mathbf{P}$.
Let $L \in \mathbf{BPP}$ (L is accepted by "clear majority").
For $|x| = n$, let $A(x) \subseteq \{0,1\}^{p(n)}$ be the set of *accepting* computations.
We have:

- $x \in L \Rightarrow |A(x)| \geq 2^{p(n)}\left(1 - \frac{1}{2^n}\right)$
- $x \notin L \Rightarrow |A(x)| \leq 2^{p(n)}\left(\frac{1}{2^n}\right)$

Let U be the set of all bit strings of length $p(n)$.
For $a, b \in U$, let $a \oplus b$ be the XOR:
$a \oplus b = c \Leftrightarrow c \oplus b = a$, so "$\oplus b$" is 1-1.

BPP and PH

Proof (cont.)

For $t \in U$, $A(x) \oplus t = \{a \oplus t : a \in A(x)\}$ (translation of $A(x)$ by t). We imply that: $|A(x) \oplus t| = |A(x)|$

If $x \in L$, consider a *random* (drawing $p^2(n)$ bits) sequence of translations: $t_1, t_2, .., t_{p(n)} \in U$.

For $b \in U$, these translations *cover* b, if $b \in A(x) \oplus t_j$, $j \leq p(n)$.

$b \in A(x) \oplus t_j \Leftrightarrow b \oplus t_j \in A(x) \Rightarrow \mathbf{Pr}[b \notin A(x) \oplus t_j] = \frac{1}{2^n}$

$\mathbf{Pr}$[b is **not** covered by any t_j]$=2^{-np(n)}$

$\mathbf{Pr}[\exists$ point that is **not** covered$] \leq 2^{-np(n)}|U| = 2^{-(n-1)p(n)}$

BPP and PH

Proof (cont.)

So, $T = (t_1, .., t_{p(n)})$ has a positive probability that it covers all of U.

If $x \notin L$,$|A(x)|$ is exp small,and (for large n) there's not T that cover all U.

$(x \in L) \Leftrightarrow (\exists T$ that cover all $U)$

So,

$$L = \{x | \exists (T \in \{0,1\}^{p^2(n)}) \forall (b \in U) \exists (j \leq p(n)) : b \oplus t_j \in A(x)\}$$

which is precisely the form of languages in $\Sigma_2\mathbf{P}$.

The last existential quantifier $(\exists(j \leq p(n))...)$ affects only polynomially many possibilities,so it doesn't "count" (can by tested in polynomial time by trying all t_j's).

□

ZPP Class

- And now something completely different:
- What is 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 $\mathcal{O}(T(|x|))$ 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:

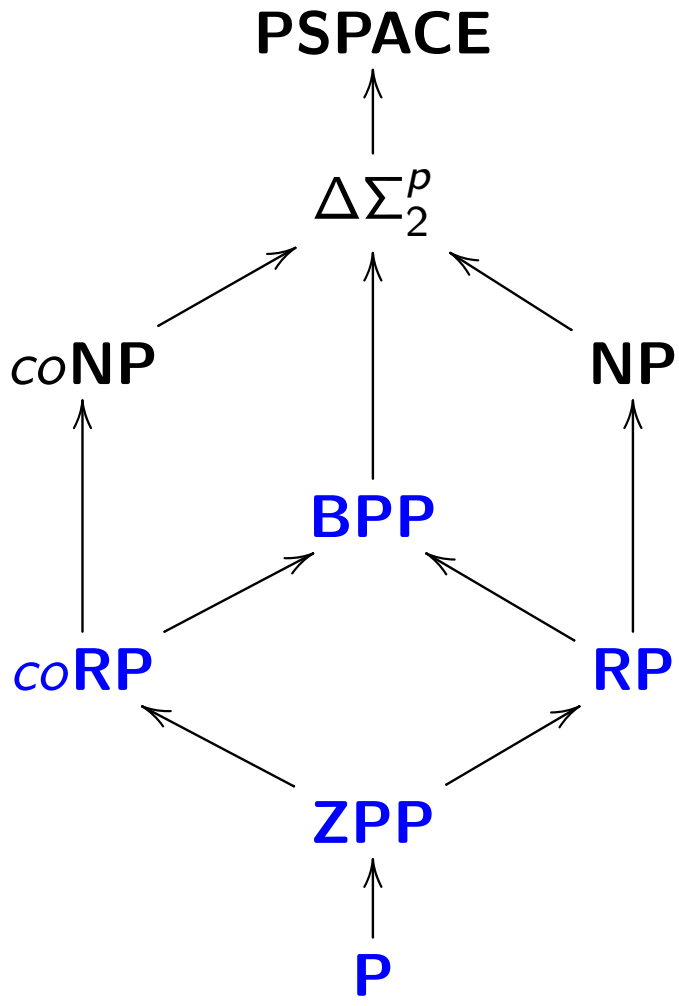
$$\mathbf{ZPP} = \bigcup_{c \in \mathbb{N}} \mathbf{ZTIME}[n^c]$$

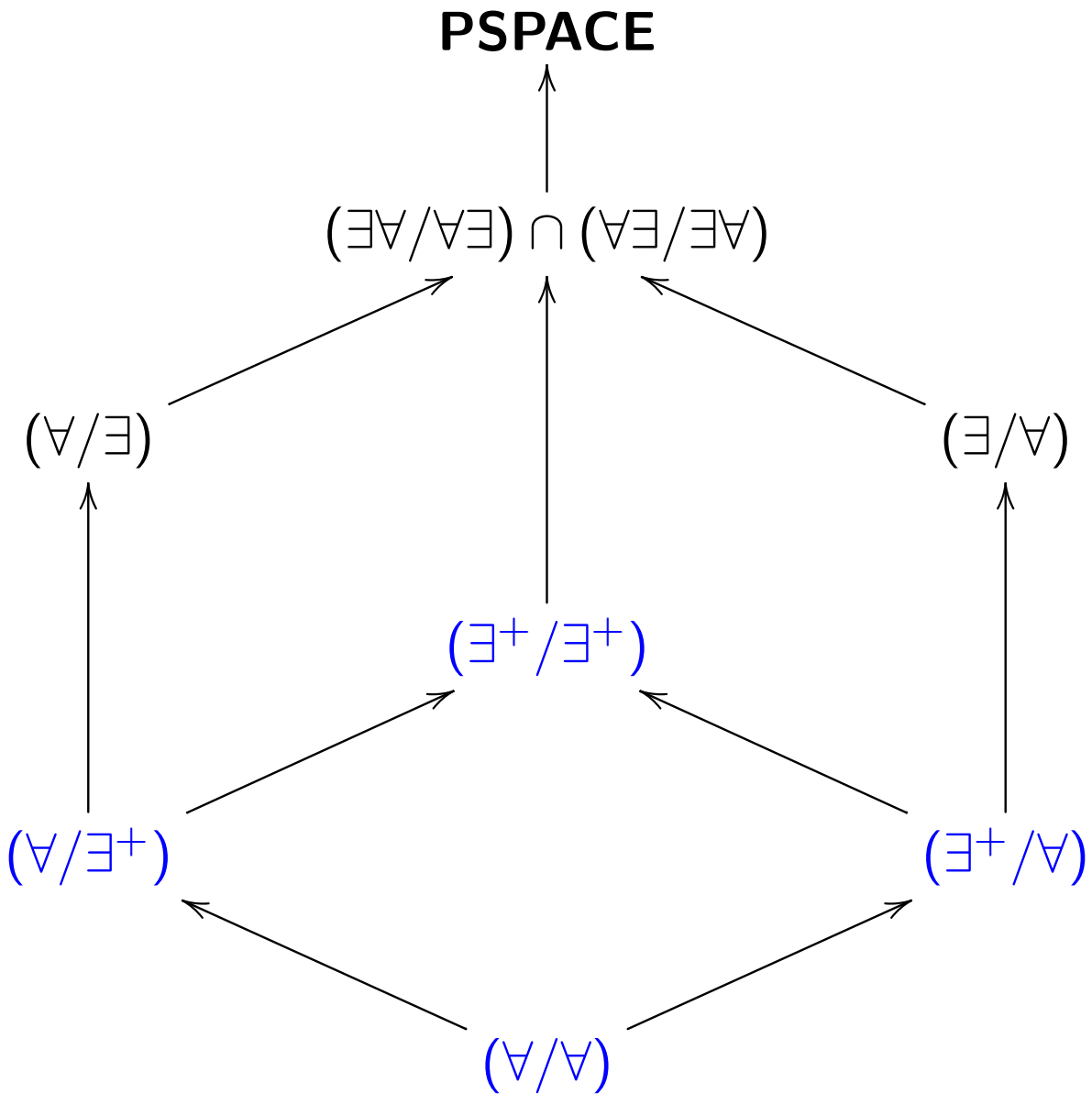
ZPP Class

- 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 $\Delta\Sigma_2^p = \Sigma_2^p \cap \Pi_2^p = \mathbf{NP}^{\mathbf{NP}} \cap co\mathbf{NP}^{\mathbf{NP}}$)

BPP and PH




PSPACE
$(\exists\forall/\forall\exists) \cap (\forall\exists/\exists\forall)$
$(\forall/\exists)$
$(\exists/\forall)$
$(\exists^+/\exists^+)$
$(\forall/\exists^+)$
$(\exists^+/\forall)$
$(\forall/\forall)$

Error Reduction for BPP

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: 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.`

We define the r.v. X_i for every $i \in [k]$ to be 1 if $y_i = L(x)$ and 0 otherwise.
$X_1, X_2, \ldots, X_k$ are indepedent Boolean r.v.'s, with:

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

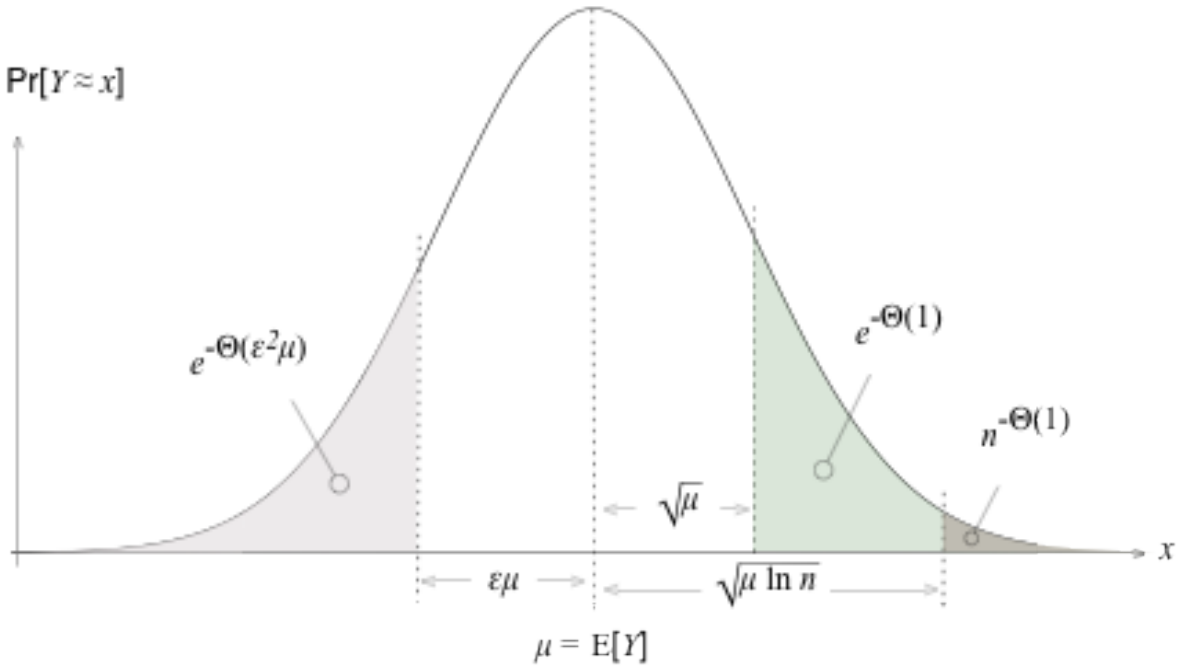
Applying a Chernoff Bound we obtain:

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

□

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

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**, $\mathbf{NP} \cap \mathit{co}\mathbf{NP}$, **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**!
- If finally **P** = **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 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!

Theorem

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

Proof:
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:

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

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, **Pr**$[M(x)$ accepts$] \geq 1/2 + 2^{-(p(n)+1)}$.
- If $x \notin A$, then **Pr**$[M(x)$ accepts$] = 1/2$.

□

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 $\mathtt{SAT} \in \mathbf{BPP}$, then $\mathtt{SAT} \in \mathbf{RP}$.
- Recall that SAT has the **self-reducibility** property: $\phi(x_1, \ldots, x_n)$: $\phi \in \mathtt{SAT} \Leftrightarrow (\phi|_{x_1=0} \in \mathtt{SAT} \ \vee \ \phi|_{x_1=1} \in \mathtt{SAT})$.
- $\mathtt{SAT} \in \mathbf{BPP}$: $\exists$ 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` ϕ `with` n `variables`
If $M(\phi) = 0$ **then** `reject` ϕ;
For $i = 1$ `to` n
$\rightarrow$ **If** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=0}) = 1$ **then** `let` $\alpha_i = 0$
$\rightarrow$ **ElseIf** $M(\phi|_{x_1=\alpha_1,\ldots,x_{i-1}=\alpha_{i-1},x_i=1}) = 1$ **then** `let` $\alpha_i = 1$
$\rightarrow$ **Else** `reject` ϕ `and halt;`
If $\phi|_{x_1=\alpha_1,\ldots,x_n=\alpha_n} = 1$ **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 $\phi \notin$ `SAT`.
- If $\phi \in$ `SAT`, then M rejects ϕ on each iteration of the loop w.p. $2^{-|\phi|}$.
- So, **Pr**[M_1 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: $\mathbf{RP}^A \neq \mathbf{NP}^A$.

Corollary

There exists an A such that:

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

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.

- 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

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

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

Theorem (Nonuniform Hierarchy Theorem)

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

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

Turing Machines that take advice

Definition

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

$$\mathbf{DTIME}\left(T(n)/a(n)\right)$$

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

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

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

Turing Machines that take advice

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

$$\mathbf{P}_{/\mathbf{poly}} = \bigcup_{c,d \in \mathbb{N}} \mathbf{DTIME}(n^c/n^d)$$

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 $L \in \mathbf{DTIME}(n^c/n^d)$. Then, since CVP is **P**-complete, we construct for every n a circuit D_n such that, for $x \in \{0,1\}^n, a_n \in \{0,1\}^{a(n)}$:

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

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

Theorem

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

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

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

- It is easy to see that $\mathtt{U_H} \in \mathbf{P}_{/\mathbf{poly}}$, but....

Theorem (Karp-Lipton Theorem)

If $\mathbf{NP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{PH} = \Sigma_2^p$.

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, $x \in L \Rightarrow \forall y \exists z\ R(x, y, z)$ Let $L \in \Pi_2^p$. Then, $x \in L \Rightarrow \forall y \underbrace{\exists z\ R(x, y, z)}_{\texttt{SAT Question}}$
- So, we can get a function $\phi(x, y) \in \mathbf{FP}$ s.t. :

$$x \in L \Leftrightarrow \forall y[\phi(x, y) \in \texttt{SAT}]$$

- Since $\texttt{SAT} \in \mathbf{P}_{/\mathbf{poly}}$, $\exists\{C_n\}_{n\in\mathbb{N}}$ s.t. $C_{|\phi|}(\phi(x, y)) = 1$ iff ϕ satisfiable.
- The idea is to nondeterministically *guess* such a circuit:

- If $x \in L$: Since $L \in \Pi_2^p$, $x \in L \Rightarrow \forall y[\phi(x,y) \in \mathtt{SAT}]$

  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 $x \notin L$: Since $L \in \Pi_2^p$, $x \notin L \Rightarrow \exists y[\phi(x,y) \notin \mathtt{SAT}]$

  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 $L \in \Sigma_2^p \Rightarrow \Pi_2^p \subseteq \Sigma_2^p$. □

Theorem (Meyer's Theorem)

If $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, *then* $\mathbf{EXP} = \Sigma_2^p$.

Theorem

$$\mathbf{BPP} \subset \mathbf{P}_{/\mathbf{poly}}$$

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:

$$\mathbf{Pr}\left[\exists x \in \{0,1\}^n : M(x,r) \neq L(x)\right] = \mathbf{Pr}\left[\bigcup_{x \in \{0,1\}^n} M(x,r) \neq L(x)\right] \leq$$

$$\leq \sum_{x \in \{0,1\}^n} \mathbf{Pr}\left[M(x,r) \neq L(x)\right] < 2^{-n} + \cdots + 2^{-n} = 1$$

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.

$\square$

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 as the size of the smallest Boolean Circuit computing f (that is, $C(x) = f(x), \forall x \in \{0,1\}^n$).

Definition (Average-Case Hardness)

The minimum S such that there is a circuit C of size S such that:

$$\mathbf{Pr}\left[C(x) = f(x)\right] \geq \frac{1}{2} + \frac{1}{S}$$

is called the (average-case) hardness of f.

Hierarchies for Semantic Classes with advice

- We have argued why we can't obtain Hierarchies for semantic measures using classical diagonalization techniques. But using small advice we can have 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 an undecidable language.
- The root of this problem lies in the "weak" definition of such families, since it suffices that $\exists$ a circuit family for L.
- We haven't a way (or an algorithm) to construct such a family.
- 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

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.

$$\mathbf{AC} = \bigcup_{i\in\mathbb{N}} \mathbf{AC}^i$$

- $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$, 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 $i \geq 0$
- $\texttt{PARITY} \in \mathbf{NC}^1$.

Circuit Lower Bounds

- The significance of proving lower bounds for this computational model is related to the famous "**P** vs **NP**" problem, since:

$$\mathbf{NP} \setminus \mathbf{P}_{/\mathbf{poly}} \neq \emptyset \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

- But...after decades of efforts, The best lower bound for an **NP** language is $5n - o(n)$, proved very recently (2005).
- There are better lower bounds for some special cases, i.e. some restricted classes of circuits, such as: bounded depth circuits, monotone circuits, and bounded depth circuits with "counting" gates.

Reminder

Let $PAR : \{0,1\}^n \to \{0,1\}$ be the *parity* function, which outputs the modulo 2 sum of an n-bit input. That is:

$$PAR(x_1, ..., x_n) \equiv \sum_{i=1}^{n} x_i (\mod 2)$$

Theorem (Furst, Saxe, Sipser, Ajtai)

$$\texttt{PARITY} \notin \mathbf{AC}^0$$

- 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

$$\mathbf{NC}^0 \neq \mathbf{AC}^0 \neq \mathbf{NC}^1$$

Definition

A language L is in $\mathbf{ACC}^0[m_1, \ldots, m_k]$ 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 $MOD_{m_1}, \ldots, MOD_{m_k}$ gates accepting L.

$$\mathbf{ACC}^0 = \bigcup_{m_1,\ldots,m_k} \mathbf{ACC}^0[m_1, \ldots, m_k]$$

- 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 $\mathbf{ACC}^0[q]$.

Theorem (Ryan Williams, 2010)

$$\mathbf{NEXP} \not\subseteq \mathbf{ACC}^0$$

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)

Denote by $CLIQUE_{k,n} : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ *the function that on input an adjacency matrix of an n-vertex graph G outputs* 1 *iff G contains an k-clique. 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 ($2^{\Omega(n^{1/8})}$).
- The importance of the above theorem lies on the fact that there was some alleged connection between monotone and non-monotone circuit complexity (e.g. that they would be polynomially related). Unfortunately, Éva Tardos proved in 1988 that the gap between the two complexities is exponential.
- 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 [Razborov, Rudich 1994]

Definition

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 $g : \{0,1\}^n \to \{0,1\}$: $M(g) = \mathcal{P}(g)$.
- For a random function g: $\mathbf{Pr}[\mathcal{P}(g) = 1] \geq \frac{1}{n}$

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 $\mathcal{P}$.

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 $\mathbf{P} = \mathbf{NP}$*, 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 $L \in \mathbf{NP}$! □

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.

- If $G_1 \ncong G_2$, then the powerfull prover can (nondeterministivally) 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 $y_i = M(x, r_1, \ldots, r_i)$ (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 $\mathbf{AM} \equiv \mathbf{AM}[2]$.

- **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

$$\texttt{GNI} \in \mathbf{AM}[2]$$

Theorem

For every $p \in poly(n)$:

$$\mathbf{IP}\left(p(n)\right) = \mathbf{AM}(p(n) + 2)$$

- So,

$$\mathbf{IP}[poly] = \mathbf{AM}[poly]$$

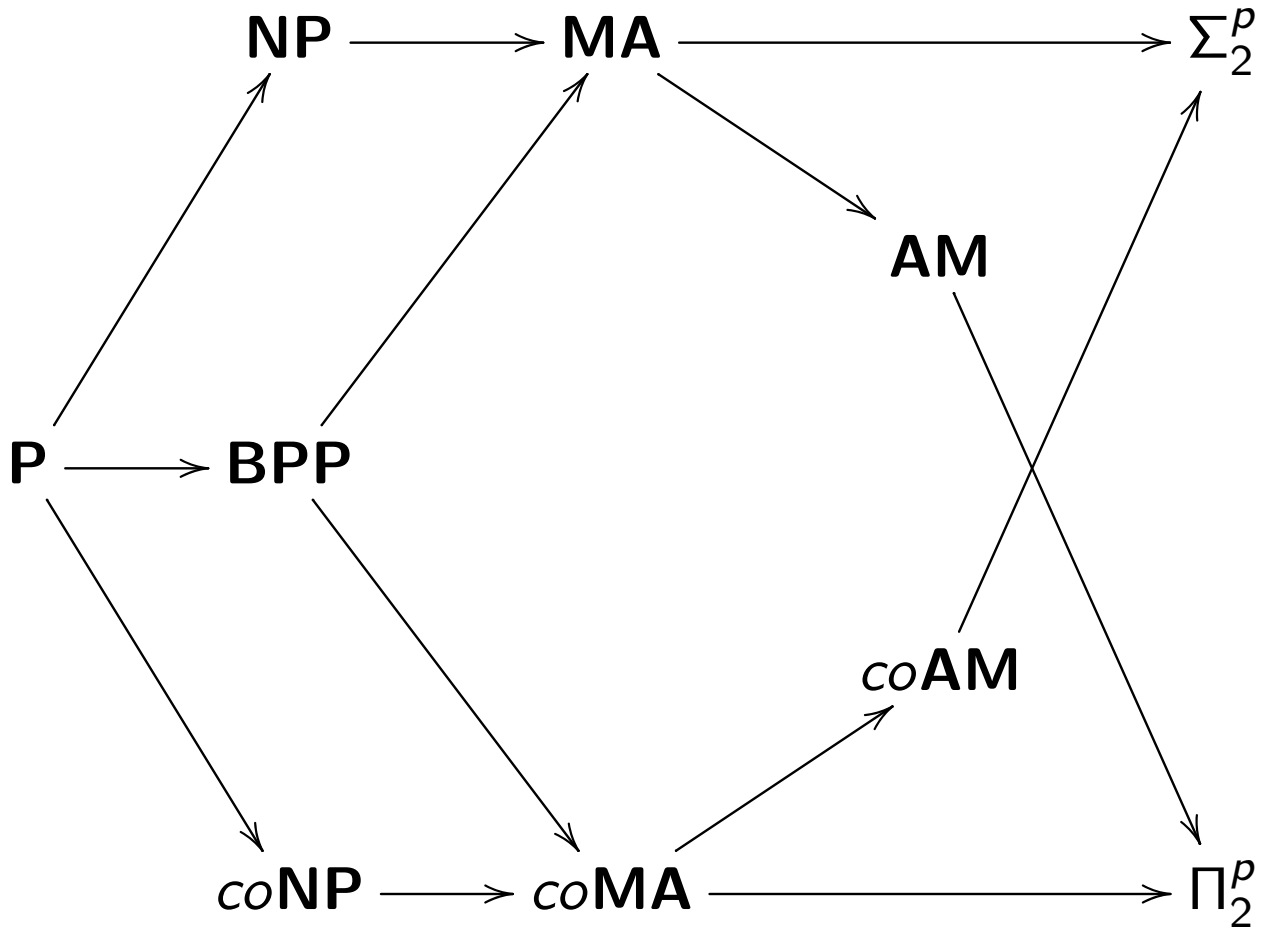
Properties of Arthur-Merlin Games

- $\mathbf{MA} \subseteq \mathbf{AM}$
- $\mathbf{MA}[1] = \mathbf{NP}$, $\mathbf{AM}[1] = \mathbf{BPP}$
- $\mathbf{AM}$ could be intuitively approached as the probabilistic version of $\mathbf{NP}$ (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



Properties of Arthur-Merlin Games

- Proper formalism (*Zachos et al.*):

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

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

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

i. $\mathbf{AM} \subseteq \Pi_2^p$
ii. $\mathbf{MA} \subseteq \Sigma_2^p \cap \Pi_2^p$

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

$$\mathbf{MAM} = (\exists\exists^{+}\exists/\forall\exists^{+}\forall) \overset{(1)}{\subseteq} (\exists\exists^{+}\forall\exists/\forall\forall\exists^{+}\forall) \subseteq (\exists\forall\exists/\forall\exists^{+}\forall) \overset{(2)}{\subseteq}$$
$$\subseteq (\forall\exists\exists/\exists^{+}\forall\forall) \subseteq (\forall\exists/\exists^{+}\forall) = \mathbf{AM}$$

Example

Properties of Arthur-Merlin Games

Proof:

- The general case is implied by the generalization of BPP-Theorem (1) & (2):
- $(\mathbf{Q_1}\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\exists^+\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) = (\mathbf{Q_1}\forall\exists^+\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ (1′)
- $(\mathbf{Q_1}\exists\forall\mathbf{Q_2}/\mathbf{Q_3}\forall\exists^+\mathbf{Q_4}) \subseteq (\mathbf{Q_1}\forall\exists\mathbf{Q_2}/\mathbf{Q_3}\exists^+\forall\mathbf{Q_4})$ (2′)
- Using the above we can easily see that the Arthur-Merlin Hierarchy collapses at the second level. (*Try it!*) □

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:

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$ $\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

$\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$ $\Sigma_2^p = (\exists\forall/\forall\exists) \overset{Hyp.}{\subseteq} (\exists\forall\exists/\forall\exists^+\forall) \overset{(2)}{\subseteq} (\forall\exists\exists/\exists^+\forall\forall) = (\forall\exists/\exists^+\forall) =$ **AM** $\subseteq (\forall\exists/\exists\forall) = \Pi_2^p$. $\square$

$\overset{Hyp.}{} \quad \overset{(2)}{}$

Measure One Results

- $\mathbf{P}^A \neq \mathbf{NP}^A$, for almost all oracles A.
- $\mathbf{P}^A = \mathbf{BPP}^A$, for almost all oracles A.
- $\mathbf{NP}^A = \mathbf{AM}^A$, for almost all oracles A.

Definition

$$almost\mathcal{C} = \left\{ L | \mathbf{Pr}_{A \in \{0,1\}^*} \left[L \in \mathcal{C}^A \right] = 1 \right\}$$

Theorem

i. $almost\mathbf{P} = \mathbf{BPP}$ *[BG81]*
ii. $almost\mathbf{NP} = \mathbf{AM}$ *[NW94]*
iii. $almost\mathbf{PH} = \mathbf{PH}$

Measure One Results

Theorem (Kurtz)

For almost every pair of oracles B, C:

i. $\mathbf{BPP} = \mathbf{P}^B \cap \mathbf{P}^C$
ii. $\mathit{almost}\mathbf{NP} = \mathbf{NP}^B \cap \mathbf{NP}^C$

Indicative Open Questions

- Does exist an oracle separating **AM** from $\mathit{almost}\mathbf{NP}$?
- Is $\mathit{almost}\mathbf{NP}$ contained in some finite level of Polynomial-Time Hierarchy?
- *Motivated by [BHZ]*: If $\mathit{co}\mathbf{NP} \subseteq \mathit{almost}\mathbf{NP}$, does it follow that **PH** collapses?

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

$$\mathbf{IP} \subseteq \mathbf{PSPACE}$$

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 prover as an **oracle**, by assuming wlog that his responses are one bit at a time.
- Th 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. □

Warmup: Interactive Proof for UNSAT

Lemma 2

PSPACE $\subseteq$ IP

- For simplicity, we will construct an Interactive Proof for `UNSAT` (a *co***NP**-complete problem), showing that:

Theorem

***co*NP $\subseteq$ 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 $(mod N)$ (where $N > 2^n \cdot 3^m$), in the following way:

Arithmetization

- Arithmetic generalization of a CNF Boolean Formula.

$$\begin{array}{rcl} \mathtt{T} & \longrightarrow & 1 \\ \mathtt{F} & \longrightarrow & 0 \\ \neg x & \longrightarrow & 1 - x \\ \wedge & \longrightarrow & \times \\ \vee & \longrightarrow & + \end{array}$$

Example

$$(x_3 \vee \neg x_5 \vee x_{17}) \wedge (x_5 \vee x_9) \wedge (\neg x_3 \vee x_4)$$
$$\downarrow$$
$$(x_3 + (1 - x_5) + x_{17}) \cdot (x_5 + x_9) \cdot ((1 - x_3) + (1 - x_4))$$

- 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
$q_1(x) = \sum p(x, x_2, \ldots x_n)$	$\longrightarrow$	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

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n) \equiv 0 \ (mod N)$$

  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 $\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n) \neq 0 \ (mod N)$

- So, $p_1(01) + p_1(1) \neq 0$, 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

$$\begin{array}{ccc} \exists & \longrightarrow & \sum \\ \forall & \longrightarrow & \prod \end{array}$$

Example

$$\forall x_1 \exists x_2 [(x_1 \wedge x_2) \vee \exists x_3 (\bar{x}_2 \wedge x_3)]$$

$$\downarrow$$

$$\prod_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left[(x_1 \cdot x_2) + \sum_{x_3 \in \{0,1\}} (1 - x_2) \cdot x_3 \right]$$

Theorem

A closed QBF is true if and only if tha value of its arithmetic form is non-zero.

Arithmetization of QBF

- If a QBF is `true`, its value could be quite large:

Theorem

Let A be a closed QBF of size n. Then, the value of its arithmetic form cannot exceed $\mathcal{O}\left(2^{2^n}\right)$.

- Since such numbers cannot be handled by the protocol, we reduce them modulo some -smaller- prime p:

Theorem

Let A be a closed QBF of size n. Then, there exists a prime p of length polynomial in n, such that its arithmetization

$$A' \neq 0 (mod p) \Leftrightarrow A \text{ is true.}$$

Arithmetization of QBF

- A QBF with all the variables quantified is called **closed**, and can be evaluated to either `True` or `False`.
- An **open** QBF with $k > 0$ free variables can be interpreted as a boolean function $\{0,1\}^k \to \{0,1\}$.
- Now, consider the language of all true quantified boolean formulas:

$$\texttt{TQBF} = \{\Phi | \Phi \text{ is a true quantified Boolean formula}\}$$

- It is known that `TQBF` is a **PSPACE**-complete language!
- So, if we have a interactive proof protocol recognizing `TQBF`, then we have a protocol for every **PSPACE** language.

Protocol for TQBF

- Given a quantified formula

$$\Psi = \forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n \ \phi(x_1, \ldots, x_n)$$

we use *arithmetization* to construct the polynomial P_ϕ. Then, $\Psi \in \texttt{TQBF}$ if and only if

$$\prod_{b_1 \in \{0,1\}^*} \sum_{b_2 \in \{0,1\}^*} \prod_{b_3 \in \{0,1\}^*} \cdots \sum_{b_n \in \{0,1\}^*} P_\phi(b_1, \ldots, b_n) \neq 0$$

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).
  (How long can be in the 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 $\exists \pi \in \{0,1\}^* : \mathbf{Pr}[V^\pi(x) = 1] = 1$
- *Soundness*: If $x \notin L$, then $\forall \pi \in \{0,1\}^* : \mathbf{Pr}[V^\pi(x) = 1] \leq \frac{1}{2}$

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:

  $\mathbf{PCP}[0, 0] = \mathbf{?P}$
  $\mathbf{PCP}[0, poly] = \mathbf{?NP}$
  $\mathbf{PCP}[poly, 0] = \mathbf{?}co\mathbf{RP}$

- A suprising result from Arora, Lund, Motwani, Safra, Sudan, Szegedy states that:

The PCP Theorem

$$\mathbf{NP} = \mathbf{PCP}[\log n, 1]$$

Main Results

- 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* solution 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.
- **FP** $\subseteq$ **#P** $\subseteq$ **PSPACE**
- If **#P** $=$ **FP**, then **P** $=$ **NP**.
- If **P** $=$ **PSPACE**, then **#P** $=$ **FP**.

- In order to formalize a notion of completeness for $\#\mathbf{P}$, we must define proper reductions:

Definition (Cook Reduction)

A function f is $\#\mathbf{P}$-complete if it is in $\#\mathbf{P}$ and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^g$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in \mathbf{NP}$, 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:

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:

$$|\{y : \ R_A(x,y) = 1\}| = |\{z : \ R_B(f(x),z) = 1\}|$$

Completeness Results

Theorem

#CIRCUIT SAT *is* #**P**-*complete.*

Proof:

- Let $f \in \#\mathbf{P}$. Then, $\exists M, p$:
  $f = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$.
- 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. □

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 $\{x_i, y_i\} \in E$ 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.*

- Notice that the decision version of PERMANENT is in **P** ! !

Proof Idea:

- We reduce 3SAT to PERMANENT in two steps:
- Given ϕ, we create an undirected graph G' with small weights, such that:

$$PERM(G') = 4^{3m} \cdot \#\phi$$

- In the second step, we convert G' to an undirected graph G such that $PERM(G) = PERM(G') \mod M$, where M has polynomially many bits.
- The problem PERMANENT MODN reduces to PERMANENT.
- Finally, the permanent of the resulting matrix is 4^{3m} times the number of sat. truth assignments of the original formula.

The Class $\oplus$P

Definition

A language L is in the class $\oplus$**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 $\oplus$**P**-complete.
- $\oplus$**P** is closed under complement.

Valiant-Vazirani Theorem

Theorem

Given a Boolean Formula F in CNF it can be constructed in polynomial time a set of formulas $F_1, F_2, \ldots, F_m$ in CNF, such that:

- *If F is satisfiable, w.p. more than $1/2$ one of F_i's is uniquely satisfiable.*
- *If F is unsatisfiable, all F_i are insatisfiable.*

The above is equivalent with:

Theorem

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

where USAT is the unique-satisfiability problem.

Valiant-Vazirani Theorem

Proof:

Definition

Let $S \subseteq \{x_1, \ldots, x_n\}$. Hyperplane η_S is a CNF Boolean Formula, s.t. an even number among the variables in S are true.

- We will construct the formulas F_i: $F_i = F_{i-1} \wedge \eta_{S_i}$, $1 \leq i \leq n+1$ where S_i is a random generated subset of th variables, and $F_0 = F$.
- If $F_i \in \texttt{USAT}$, then we answer that F is satisfiable.
- If none of the F_i's are in $\texttt{USAT}$, then we answer that F is probably unsatisfiable.
- We shall prove now that the probability to be wrong is $< 7/8$ (and by repeating the algorithm 6 times we are $< 1/2$, as required:

Valiant-Vazirani Theorem

Lemma

If the number of satisfying truth assignments of F is between 2^k and 2^{k+1}, $0 \leq k < n$, then the probability that F_{k+2} has exactly one satisfying truth assignments is at least $1/8$.

Proof (of the Lemma):

- Let T the number of satisfying t.a.'s.
- Two t.a. **agree** on η_S if they both satisfy or falsify it.
- Fix $t \in T$.
- A $t' \in T$ agrees with t on all $k+2$ first hyperplanes w.p. $\frac{1}{2^{k+2}}$.
- Summing up over all $t' \in T \setminus t$, t agrees with *some* t' on the first $k+2$ hyperplanes w.p. at most $\frac{|T|-2}{2^{k+2}}$.
- So, the probability that t **disagrees** with eery other member of T on one of the first $k+2$ hyperplanes is at least $1/2$.

Valiant-Vazirani Theorem

- The probability that t satisfies all $k+2$ first hyperplanes is $\frac{1}{2^{k+2}}$, and with probability $1/2$ it is te only one that does.
- So, with probability at least $\frac{1}{2^{k+3}}$ t is the unique satisfying t.a. of F_{k+2}.
- Since this holds for each $t \in T$, the probability that such an element of T exists is $2^k \times \frac{1}{2^{k+3}} = \frac{1}{8}$. □

Proof (cont'd):

- If the number of satisfying t.a.'s of F is not zero, then it lies between 2^k and 2^{k+1}, for some $k < n$.
- So, at least one of the F_i will have probability at least $1/8$ to be satisfied by a unique t.a. □

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)

$$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$$

Toda's Theorem

- The proof consists of two main lemmas:

Lemma 1

$$\mathbf{PH} \subseteq \mathcal{BP} \cdot \oplus\mathbf{P}$$

Lemma 2

$$\mathcal{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\#\mathbf{P}}$$

Operators Reminder

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:

$$x \in L \Leftrightarrow |\{y : \ |y| = p(|x|), x; y \in A\}| \text{ is odd}$$

- $\oplus \cdot \mathbf{P} = \oplus\mathbf{P}$

Definition

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:

- $x \ \in L \Rightarrow \exists y : \ x; y \in A$
- $x \ \notin L \Rightarrow \forall y : \ x; y \in A$

- $\mathcal{N} \cdot \mathbf{P} = \mathbf{NP}$
- $\mathcal{N} \cdot \Pi^p_{i-1} = \Sigma^p_i$

Operators Reminder

Definition

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:

- $x \in L \Rightarrow \exists^{+} y : \ x; y \in A$
- $x \notin L \Rightarrow \exists^{+} y : \ x; y \in A$

- $\oplus \cdot \oplus \cdot \mathbf{C} = \oplus \cdot \mathbf{C}$

Toda's Theorem

Lemma 1.1

$$\oplus \cdot \mathcal{BP} \cdot \mathbf{C} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{C}$$

Proof Idea:

- If $A \in \mathcal{BP} \cdot \mathbf{C}$, then there is a $B \in \mathcal{C}$ such that:
  $y \in A \Rightarrow \mathbf{Pr}_w[y; w \notin B] \leq 2^{-(|y|+2)}$
  $y \notin A \Rightarrow \mathbf{Pr}_w[y; w \in B] \leq 2^{-(|y|+2)}$
- So, $\mathbf{Pr}_w[\exists y, |y| = n, y; w \in B \Leftrightarrow y \notin A] \leq$
  $\sum_{|y|=n} \mathbf{Pr}_w[y; w \in B \Leftrightarrow y \notin A] \leq 2^n \cdot 2^{-(n+2)} = \frac{1}{4}$
- Let $L \in \oplus \cdot \mathcal{BP} \cdot \mathbf{C}$, then $\exists A \in \mathcal{BP} \cdot \mathbf{C}$:

$$x \in L \Leftrightarrow |\{z \mid |z| = |x|^k \wedge x; z \in A\}| \text{ is odd}$$

Toda's Theorem

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:

$$\Sigma_k^p = \mathcal{N} \cdot \Pi_{k-1} \subseteq \mathcal{N} \cdot \mathcal{BP} \cdot \oplus \cdot \mathbf{P} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$$

$$\subseteq \mathcal{BP} \cdot \mathcal{BP} \cdot \oplus \cdot \oplus \cdot \mathbf{P} \subseteq \mathcal{BP} \cdot \oplus \cdot \mathbf{P}$$

□

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:

$$\mathbf{Pr}_y[x \in L \Leftrightarrow x; y \in A] \geq \frac{3}{4}$$

Lemma 2.1

For $A \in \oplus\mathbf{P}$, and $\forall q \in poly(n)$, $\exists M$:

- $x \in A \Rightarrow \#acc_M(x) \equiv -1 \mod 2^{q(n)}$
- $x \notin A \Rightarrow \#acc_M(x) \equiv 0 \mod 2^{q(n)}$

Toda's Theorem

- By Lemma 2.1:
  - $x; y \in A \Rightarrow \#acc_M(x) \equiv -1 \mod 2^{p(n)}$
  - $x; y \notin A \Rightarrow \#acc_M(x) \equiv 0 \mod 2^{p(n)}$
- Let $g(x) = |\{y \mid |y| = p(|x|), x; y \in A\}|$
- Let $h(x) = \sum_{x;y \in A} \#acc_M(x; y) + \sum_{x;y \notin A} \#acc_M(x; y) = \cdots \equiv -g(x) \mod 2^{p(n)}$.
- So, we can decide $x \in L$ from $h(x)$.