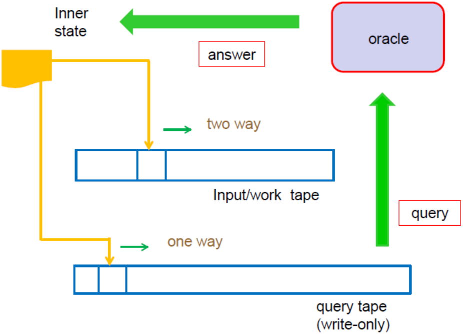
Oracle Computation/Polynomial-time Hierarchy

Oracle Turing Machines

Definition 1

An oracle for a language A answers whether $w \in A$ for any string w. An oracle Turing machine M^A is a Turing machine that can query an oracle A. When M^A write a string w on a special oracle tape, it is informed whether $w \in A$ in a single step.



Oracle Computations

- Let M be an oracle Turing machine (OTM)
- Let x be any string in Σ^*
- Let B be an oracle (which is now a language).
  1. M starts with input x.
  2. Whenever M writes a query word y on its query tape and enters a query state q_{query}, y is automatically sent to oracle B.
  3. The oracle B returns its answer (YES/NO) by changing M's inner state from q_{query} to either q_{yes} or q_{no}, depending on whether $y \in B$ or $y \notin B$, respectively.
  4. M resumes its computation, starting with q_{yes} or q_{no}.

Definition 2

$L(M^B) = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ with oracle } B\}$.

Oracle Turing Machines

Definition 3

For two languages A and B, we say that A is Turing reducible to B (written as $A \leq_T B$) if there is an OTM M such that

1. $A = L(M^B)$; that is, for every input x, $x \in A \Leftrightarrow M^B$ accepts x via making queries to the oracle B

Definition 4

Language A is polynomial-time Turing reducible to language B (written as $A \leq_T^p B$ if there is an OTM M such that

1. $A = L(M^B)$; that is, for every input x, $x \in A \Leftrightarrow M^B$ accepts x via making queries to the oracle B
2. M runs in polynomial time.

Polynomial-time Oracle Turing Machines

Definition 5

$P^A = \{L : L \text{ is decided by a polynomial time OTM with oracle } A\}$
$NP^A = \{L : L \text{ is decided by a polynomial time ONTM with oracle } A\}$

Example 6

$NP \subseteq P^{SAT}$ and $coNP \subseteq P^{SAT}$.

Proof.

For any $A \in NP$, use the polynomial reduction of A to SAT. $\square$

Oracle Turing Machines

- Two Boolean formulae ϕ and ψ over $x_1, \ldots, x_l$ are equivalent if they have the same value on any assignments to $x_1, \ldots, x_l$.
- A formula is minimal if it is not equivalent to a smaller formula.
- Consider

$$NONMINFORMULA = \{\langle\phi\rangle : \phi \text{ is not a minimal Boolean formula}\}.$$

Example 7

$NONMINFORMULA \in NP^{SAT}$.

Proof.

"On input $\langle\phi\rangle$:

1. Nondeterministically select a smaller formula ψ.
2. Ask $\langle\phi\ XOR\ \psi\rangle \in SAT$.
3. If yes, accept; otherwise, reject."

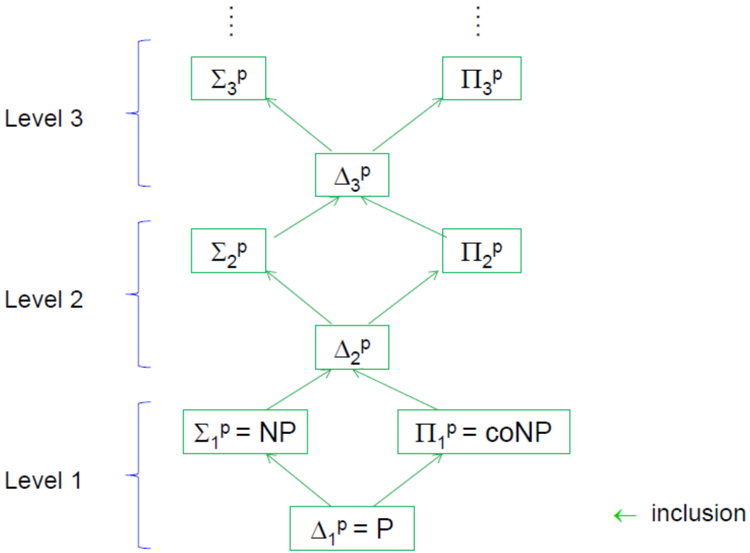
□

Polynomial-time Hierarchy

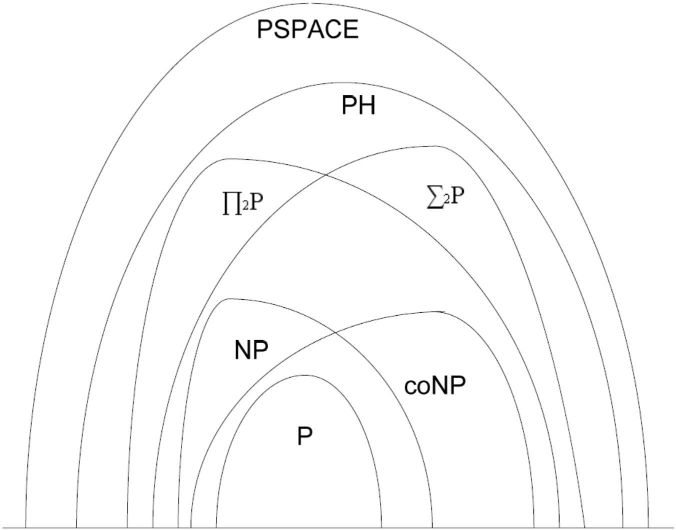
Meyer and Stockmeyer (1972, 1973) introduced a notion of the polynomial-time hierarchy over NP.
The polynomial hierarchy consists of the following complexity classes: for every index $k \geq 1$,

1. $\Delta_1^P = P$
2. $\Sigma_1^P = NP, \;\; \Pi_1^P = co\text{-}NP$
3. $\Delta_{k+1}^P = P^{\Sigma_k^P}$
4. $\Sigma_{k+1}^P = NP^{\Sigma_k^P}, \;\; \Pi_{k+1}^P = co\text{-}\Sigma_{k+1}^P$

Polynomial-time Hierarchy



Polynomial-time Hierarchy



Polynomial-time Hierarchy

We define the complexity class *PH* as follows:

$$PH = \bigcup_{k \geq 1} (\Sigma_k^P \cup \Pi_k^P)$$

- $NP \subseteq PH \subseteq PSPACE$
- If $P = NP$, then $P = PH$.
- $P^{PH} = NP^{PH} = PH$.

Another Characterization of Polynomial-time Hierarchy

We have already seen, that deciding whether a formula is satisfiable

- $\exists x_1 \cdots x_n (x_1 \vee \bar{x_2} \vee x_8) \wedge \cdots \wedge (\bar{x_6} \vee x_3)$
  - only existential quantifier – NP-complete
- $\exists x_1 \forall x_2 \exists x_3 \ldots (x_1 \vee \bar{x_2} \vee x_8) \wedge \cdots \wedge (\bar{x_6} \vee x_3)$
  - existential & universal quantifiers – PSPACE-complete

Definition 8

Consider language classes reducible to deciding the satisfiability of

$$\Sigma_i SAT : \exists x_1 \forall x_2 \exists x_3 \ldots R(x_1, x_2, x_3 \ldots)$$

$$\Pi_i SAT : \forall x_1 \exists x_2 \forall x_3 \ldots R(x_1, x_2, x_3 \ldots)$$

with i alternating quantifiers and $R(\ldots)$ is a polynomial-time predicate.

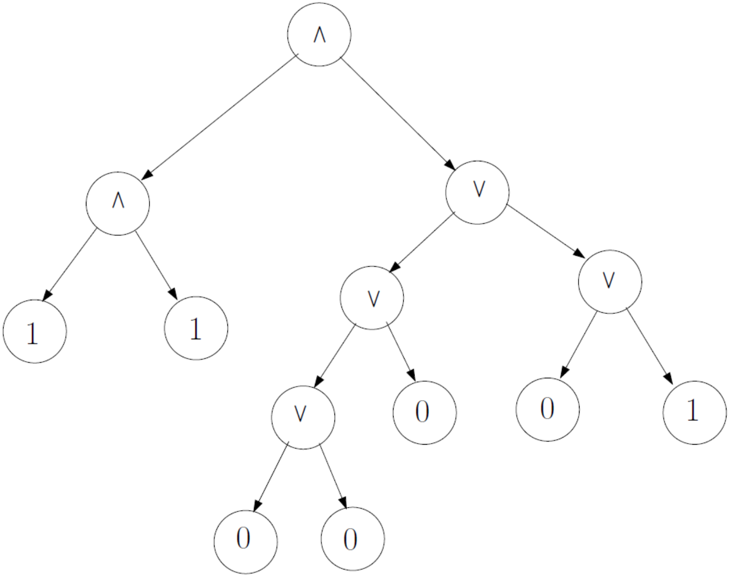
$\Sigma_i SAT$ and $\Pi_i SAT$ above define exactly the i-level of the polynomial-time hierarchy using polynomial-time oracle TMs.

Alternating Turing Machines

- An alternating Turing machine (ATM) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a Turing machine with a non-deterministic transition function $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$ whose set of states, in addition to accepting/rejecting states, is partitioned into existential ($\exists$ or $\vee$) and universal ($\forall$ or $\wedge$) states.
- A configuration C of an ATM M can reach acceptance if either of the following is true:
  - C is existential and some branch can reach acceptance.
  - C is universal and all branches can reach acceptance.

  M accepts a word w if the start configuration on w is accepting.

Alternating Turing Machines



Alternating Polynomial-time Hierarchy

Definition 9

Consider language classes

- $A\Sigma_i^p$: the language accepted by polynomial time ATM using at most i alternations with the initial state an $\exists$-state,
- $A\Pi_i^p$: the language accepted by polynomial time ATM using at most i alternations with the initial state an $\forall$-state,

It turns out that $A\Sigma_i$ and $A\Pi_i$ above again define exactly the i-level of the polynomial-time hierarchy using polynomial-time oracle TMs.

More on Alternating Complexity Classes

We define

- APTime $= \bigcup_{d\geq 1} ATime(n^d)$
- AExpTime $= \bigcup_{d\geq 1} ATime(2^{n^d})$
- ALogSpace $= \bigcup_{d\geq 1} ASpace(\log n)$
- APSpace $= \bigcup_{d\geq 1} ASpace(n^d)$
- AExpSpace $= \bigcup_{d\geq 1} ASpace(2^{n^d})$

Theorem 10

$$\begin{array}{ccccccccc} \mathsf{L} & \subseteq & \mathsf{PTime} & \subseteq & \mathsf{PSpace} & \subseteq & \mathsf{ExpTime} & \subseteq & \mathsf{ExpSpace} \\ & & \parallel & & \parallel & & \parallel & & \parallel \\ & & \mathsf{ALogSpace} & \subseteq & \mathsf{APTime} & \subseteq & \mathsf{APSpace} & \subseteq & \mathsf{AExpTime} \end{array}$$

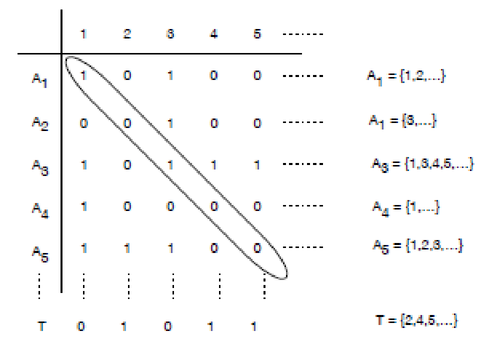
Diagonalization - Cantor's Argument

Recall Cantor's Argument for showing $2^{\mathbb{N}}$ is not countable

Proof.

Suppose for a contradiction that $2^{\mathbb{N}}$ is countable.

- Then the sets in 2^{S} can be enumerated in a list $A_1, A_2, A_3, \ldots \subseteq S$
- For a contradiction, define a set $T = \{i \mid i \in N, i \notin A_i\}$.



Diagonalization - General Idea

- Given a string $\alpha \in \{0,1\}^*$, let M_α be the TM with encoding α.
- Consider the function $f : \{0,1\}^* \to \{0,1\}$ defined by
  - $f(\alpha) = 1$ if $M_\alpha(\alpha) = 0$;
  - $f(\alpha) = 0$ if $M_\alpha(\alpha) = 1$

Theorem 11

No Turing machine can compute $f(\alpha)$.

Proof.

Note that $M_\alpha(\alpha) = f(\alpha)$. However, $f(\alpha) = 1$ (resp., 0) implies $M_\alpha(\alpha) = 0$ (resp., 1) - a contradiction. $\square$

Applications of the Diagonalization Method

- **The Halting Problem**:
  Define $M_\alpha(x)$ as $HALT(\alpha, x)$ (=1, if M_α halts on x; =0, otherwise).
  Consider $f(\alpha) = M_\alpha(\alpha)$.
- **Space Hierarchy Theorem**:
  Define function $f : \{0,1\}^* \rightarrow \{0,1\}$:
  - $f(\alpha) = 1$ if $M_\alpha(\alpha)$ halts and outputs 0 using at most $s(n)$ space;
  - $f(\alpha) = 0$ otherwise.

  (Claim 1): f can be computed in $O(s(n))$ space.
  (Claim 2): f cannot be computed in $o(s(n))$ space.
- **Time Hierarchy Theorem**: Similar to the space hierarchy theorem.

Applications of the Diagonalization Method

- **Gödel Incompleteness theorem**: "Every consistent finite set of axioms is incomplete."
  - Let $K(x)$, $x \in \{0,1\}^*$, be the length of the shortest TM M_α on blank tape that outputs x.
  - For each $x \in \{0,1\}^*, N \in \mathbb{N}$, define $S_{x,N}$ as "$K(x) > N$".
  - FACT: For every $N \in \mathbb{N}$, there exists an $x \in \{0,1\}^*$, $S_{x,N}$ holds.
    - ★ (Reason): For every $N \in \mathbb{N}$, the number of TMs (of length $\leq N$) is finite. Hence, there are only a finite number of x for which $K(x) \leq N$.
  - Given a finite set of axioms A, consider TM M_N:
    - ★ Enumerate all (x, α), $x, \alpha \in \{0,1\}^*$, if α describes a proof of $S_{x.N}$ using A, output x.
  - If A is complete, M_N always holds and outputs x, for every x. Note that $|M_N| = O(\log N)$ (using binary encoding).
    - ★ What the above says is that for every x, the shortest TM generating x is of length $\leq \log N$, which contradicts the "proof".

Limits of the Diagonalization Method

- We have seen many applications of the diagonalization methd.
  - Particularly, the proofs of space and time hierarchy theorems.
- Can we use the diagonalization method to show $P \stackrel{?}{=} NP$?
  - Say, to construct an NTM that accepts $\langle M \rangle 10^n$ if and only if the polynomial time TM M rejects $\langle M \rangle 10^n$.
- We give a strong evdience to explain why it may not work.
- The diagonalization method basically simulates a TM M by a TM D. If M and D are given an oracle A, D^A can simulate M^A as well.
- Hence if the diagonalization method can prove $P \stackrel{?}{=} NP$, it can also prove $P^A \stackrel{?}{=} NP^A$ for any oracle A.
- We will now give two oracles A and B such that $P^A \neq NP^A$ and $P^B = NP^B$.
- The diagonalization method does not suffice to prove $P \stackrel{?}{=} NP$.

Limits of the Diagonalization Method

Theorem 12

There are oracles A and B such that $P^A \neq NP^A$ and $P^B = NP^B$.

Proof.

Let B be $TQBF$. Then $NP^{TQBF} \subseteq NPSPACE \subseteq PSPACE \subseteq P^{TQBF}$.
For any oracle C, define

$$L_C = \{1^n : \exists x \in C\ [\ |x| = n\ ]\}.$$

Clearly, $L_C \in NP^C$ for any C. We construct a language A such that $L_A \notin P^A$.

$\exists A, NP^A \not\subseteq P^A$

Proof.

- Let $M_1^?, M_2^?, \ldots$ be an enumeration of oracle DTMs that run in polynomial time. Assume for simplicity that $M_i^?$ has running time n^i. Since oracle machines query their oracle as a black box, can plug in any oracle.
- We will build an oracle A so that none of these machines can decide L_A.
- Inductive construction. We start with nothing, and at each stage we declare a finite set of strings to be in the language of A or out of it.
- Goal: At stage i, make sure that $L(M_i^A)$ and L_A disagree on some string.

$\square$

$\exists A, NP^A \not\subseteq P^A$

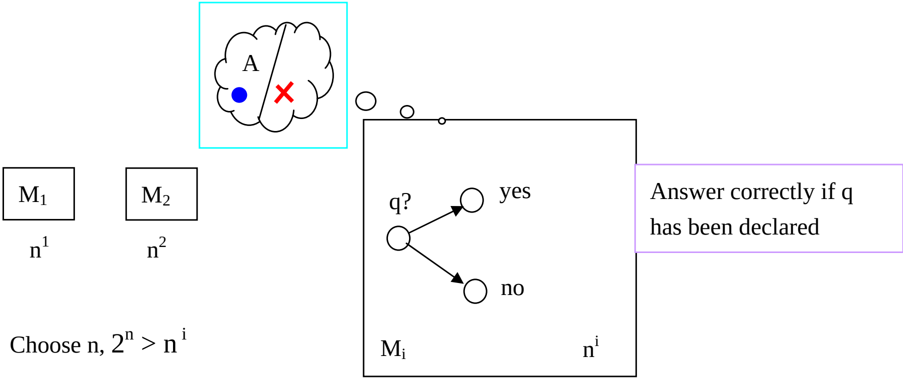
Proof.

- **Stage** i
  - Let M_i^A have running time n^i . Choose n larger than any string declared for A, such that $2^n > n^i$.
  - We are going to run M_i^A on 1^n. When M_i^A queries A with q, we
    - Answer correctly if q has been declared,
    - and answer NO otherwise.
  - If M_i^A accepts 1^n, we declare all strings of length n to be NO-strings. Then A has no YES-string of length n, and $1^n \notin L_A$. .
  - If M_i^A rejects 1^n, we find a string of length n that M_i^A did not query. This exists, since $2^n > n^i$. Declare this string to be YES.
- Finally, declare all undeclared strings of length up to n arbitrarily.

Hence M_i accepts $\mathtt{1}^n$ if and only if $\mathtt{1}^n \notin L_A$. M_i does not decide L_A.

□

$\exists A, NP^A \not\subseteq P^A$



Choose n, $2^n > n^i$

- M_i accept 1^n, declare all strings of length n to be NO-strings
- M_i rejects 1^n, we find a string w length n not queried by M_i and adds w to A