

Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lect IV

Instructor: Goutam Biswas *Autumn Semester 2026*

1 Diagonalization

We know that the following language is Turing *recognizable* (*recursively enumerable*) but not Turing *decidable* (*recursive*).

$$H_{TM} = \{< M, x >: \text{ the DTM } M \text{ accepts the input } x\}.$$

The proof was by reduction to a contradiction using *diagonalization*.

Diagonalization is also a tool to show that some complexity classes are not the same. It is used to prove *hierarchy theorems* of resource-bounded Turing machines. If a Turing machine is given more computational resources such as *time* or *space*, it can solve more problems.
Diagonalization is also used to prove an interesting theorem that, if $\mathbf{P} \neq \mathbf{NP}$, there are problems that are neither in **P** nor **NP**-*complete*.

Researchers also realized the limitations of diagonalization in addressing the **P** verses **NP** question. The limitation of diagonalization is also proved by diagonalization.

We note the following facts used in a diagonalization proof.

- Each $x \in \{0,1\}^*$ represents a Turing machine[1] M_x.
- A universal Turing machine U can simulate a Turing machine M_x (represented by $x \in \{0,1\}^*$), with a logarithmic overhead. If a Turing machine M_x halts on an input w within $O(f(n))$ time, then its simulation by an universal Turing machine $\mathcal{U}$ halts within $O(f(n) \log f(n))$ time.
- If a language L is described by a Turing machine M, then there are infinitely many Turing machines for the same language (equivalent Turing machines).

[1] We may associate a fixed useless machine to each string that actually does not encode a machine.

Allowing more computational time to a Turing machine model can enlarge the set of languages that can be decided. Consider the function $f : \mathbb{N}_0 \to \mathbb{N}_0$. The function is non-decreasing and time-constructible. Given an input x of length n, $f(n)$ can be computed in $O(f(n))$ time, and the value of $f(n)$ does not decrease with the increase in the length of the input. We have already defined the language class DTIME($f(n)$) as the collection of languages decided by a Turing machine in $O(f(n))$ time.

Theorem 1. (*Time Hierarchy Theorem*)
If $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ are *time constructible* functions and $f(n) \log f(n) = o(g(n))$, then

$$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{DTIME}(g(n))$$

Proof: Here is a proof for a simpler result,

$$\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^{1.5})$$

Consider the following DTM:
D : input x

(i) The Universal Turing machine $\mathcal{U}$ runs for $|x|^{1.4}$ steps to simulate M_x (the DTM encoded by x) on x.

(ii) If the simulation halts with $M_x(x) = 0$ (*reject*), then output 1 (*accept*).

(iii) If the simulation halts with $M_x(x) = 1$ (*accept*) or does not halt within this time, then the simulation halts with output 0 (*reject*).

As the DTM D halts within $O(|x|^{1.4}|)$ time, $L(D) \subseteq \mathbf{DTIME}(n^{1.5})$.
We claim that $L(D) \notin \mathbf{DTIME}(n)$.

Suppose, for the sake of contradiction, there is a DTM M that decides $L(D)$ in linear time i.e. given an input x, M halts in time $c|x|$, where c is a constant, and for all $x \in \{0,1\}^*$, $M(x) = D(x)$.

A Universal Turing machine $\mathcal{U}$ can simulate M on input x in time $c_1 c|x| \log |x|$. The constant c_1 depends on M and not on x.
But then there is some n_0 such that for all $n \geq n_0$, $n^{1.4} > c_1 cn \log n$.
There are infinitely many x representing the same Turing machine, i.e., $L(M_x) = L(M)$.
We take one such x so that $|x| \geq n_0$. We run D on this x. The simulation of M_x on x will certainly halt as $|x|^{1.4} > c_1 c|x| \log |x|$. But then $D(x) = 1$ if and only if $M_x(x) = M(x) = D(x) = 0$ - a contradiction. QED.

Theorem 2. (*Space Hierarchy Theorem*) Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a *space constructible* function. There is a language L that is decidable in $O(f(n))$ space but not in $o(f(n))$ space.

The language L will be described by designing an $O(f(n))$ space-bounded DTM D. The TM D is so designed that L is different from every language decidable in $o(f(n))$ space.

If M is a decider running in $o(f(n))$ space, then $L(D) = L$ is different from $L(M)$ in at least one $x \in \{0,1\}^*$. This is the diagonalization.

The *asymptotic* behavior sets in beyond some length of the input. The TM M may be $o(f(n))$ space bounded, but below the input length n_0, for the finite number of inputs, the space requirement may exceed $f(n)$. Given the input

$< M >$, D simulates M on $< M >$. But the work-space $f(| < M > |)$ in D may not be sufficient to complete the simulation of M on $< M >$. So the input to D is padded with 01^* to give enough space to complete the simulation of M.

Proof. The following $O(f(n))$ space bounded DTM D decides a language $L(D) = L$ that is not decidable in $o(f(n))$ space.
$D =$ input x

(i) Let $|x| = n$.

(ii) Compute $f(n)$ (space constructible). Mark $f(n)$ number of cells on the work-tape. Halt with 0 (*reject*) if the computation goes beyond the marked end.

(iii) If $x \neq < M > 01^*$ for some DTM M, halt with 0 (*reject*).

(iv) Simulate M on $x =< M > 01^*$ for at most $2^{f(n)}$ steps. If it goes beyond this, halt with 0 (*reject*). The number of configurations cannot be more than $2^{f(n)}$ with $f(n)$ space[2].

(v) If M halts with $M(x) = 1$, then halt with 0 (*reject*); otherwise halt with 1 (*accept*).

By definition $L(D) = L$ is decided in $O(f(n))$ space. Following argument establishes that L cannot be decided in $o(f(n))$ space.
Suppose, for the sake of argument, there is a $g(n) = o(f(n))$ space bounded DTM M so that $L = L(M)$. Let $\forall n \geq n_0$, $dg(n) \leq f(n)$. On the input $x =< M > 01^n$, $n \geq n_0$, the simulation of M on $x =< M > 01^n$ will be over in stage **(v)**. And

$$M \text{ accepts } x \text{ if and only if } D \text{ rejects } x.$$

It is a contradiction of the assumption that $L(D) = L(M)$.

1.1 NP Intermediate Language

It is known that $\mathbf{P} \subseteq \mathbf{NP}$ and there are a large number of problems that are **NP**-complete. It is not known whether $\mathbf{P} = \mathbf{NP}$. But there is a strong *belief* that they are not equal. If $\mathbf{P} \neq \mathbf{NP}$, then the question is whether there is a problem in **NP** that in not in **P** and also not **NP**-*complete*. The following theorem of Ladner ([LAD75]) gives an answer.
Theorem 3. If $\mathbf{P} \neq \mathbf{NP}$, then there is a language $L \in \mathbf{NP} \setminus \mathbf{P}$ that is not **NP**-complete.

Consider the following language defined for all $H : \mathbb{N}_0 \to \mathbb{N}_0$ and all satisfiable Boolean formulae of length $n \in \mathbb{N}_0$, padded with $n^{H(n)}$ 1s.

$$SAT_H = \{\phi\ 0\ 1^{n^{H(n)}} :\ \phi \in SAT \text{ and } |\phi| = n\}.$$

We shall define the function $H : \mathbb{N}_0 \to \mathbb{N}_0$. But before that, we have the following observation about the modification of SAT to SAT_H using the *padding* of strings of 1s at the end. If $H(n) = c$, a constant for all n, the *padding* is of polynomial length, and SAT_H is **NP**-complete. SAT can be reduced to

[2]The encoding of the alphabet and state of M on D may require $dg(n)$ space where $g(n)$ is the space bound of M.

SAT_H in polynomial time, $\phi \mapsto \phi 01^{n^c}$, copy ϕ and pad it with n^c number of 1's separated by a zero.
But if $H(n) = n$, then $SAT_H \in \mathbf{P}$. The reason is that the *satisfiability* of ϕ can be tested using a *truth table*. Though the size of the *truth-table* is exponential in the size of ϕ, it is polynomial (in fact linear) in the length of $\phi 01^{n^n}$. In fact ϕ cannot be reduced to $\phi 01^{n^n}$ in polynomial time for a large n.

The function H is defined in such a way that it is between these two extreme conditions. It is not a constant function; it grows slowly so that $\lim_{n\to\infty} H(n) = \infty$. SAT_H is neither **NP**-complete, nor in **P** under the assumption $\mathbf{P} \neq \mathbf{NP}$.

$H(n)$ is the smallest number $i \leq \log\log n$ such that for every $x \in \{0,1\}^*$ and $|x| \leq \log n$, the Turing machine M_i (encoded by i) decides the membership of x in SAT_H ($M_i(x) = 1$ if and only if $x \in SAT_H$) within $i \times |x|^i$ steps.
If there is no such i, then $H(n) = \log\log n$.

We may put $H(0) = H(1) = 1$. The membership question of SAT_H requires the computation of $H(n)$, and the value of $H(n)$ depends on the the membership of strings in SAT_H. Nevertheless, H is well defined as it tests the membership of strings of lengths greater than n, but relies upon the membership status of strings of length $\leq \log n$.

Example 1. Let $|\phi| = 2^{2^k} = n$, The length of $x = \phi 01^{n^{H(n)}}$ is longer than n. The value of $H(n) \leq \log\log(2^{2^k}) = k$ i.e. $H(n) = O(\log\log n$. The maximum possible length of $\phi 01^{n^{H(n)}}$ is $n + 1 + n^k$. We run Turing machines M_i corresponding to $1 \leq i \leq k$, on inputs from $\{x \in \{0,1\}^* : \ |x| \leq 2^k\}$, for $i \times |x|^i$ steps.

Exercise 1. Let $L = \lfloor \log_2 n \rfloor$ and $K = \lfloor \log_2 \log_2 n \rfloor$. Answer the following questions.

(i) What is the upper bound of $H(n)$?

(ii) H is a non-decreasing function?

(iii) What is the maximum number of TMs to simulate to compute $H(n)$?

(iv) On how many inputs does, each machine run?

(v) What is the upper bound of time to check the membership of ϕ in SAT for all input of length $m \leq L = \lfloor \log_2 n \rfloor$?

(vi) What is the cost of recursive calls?

(vii) Give an upper bound of computation time of $H(n)$.

Ans.

(i) The upper bound of $H(n)$ is $\lfloor \log\log n \rfloor$.

(ii) Let $m < n$, $H(m) = k$ and $H(n) = l$.
By the definition of H, either k is the smallest integer $\leq \log\log m$, such that the TM M_k correctly predicts the membership of all input x of length $\leq \log m$ in time $k \times |x|^k$, otherwise $k = \log\log m$.
Similarly, either l is the smallest integer $\leq \log\log n$, such that the TM M_l correctly predicts the membership of all input x of length $\leq \log n$ in time $l \times |x|^l$, otherwise $l = \log\log m$.

$m < n \Rightarrow \log m < \log n \Rightarrow \{x \in \{0,1\}^* : |x| < \log m\} \subseteq \{x \in \{0,1\}^* : |x| < \log n\}$. If $l < k$ and M_l decides both the sets, then $H(m) = l$, a contradiction.

(iii) The number of Turing machines simulated to compute the value of $H(n)$ is $K = \lfloor \log_2 \log_2 n \rfloor$.

(iv) The total number of input to each machine is the number of strings of lengths at most $L = \log_2 n$. Number of such strings are $1 + 2 + 2^2 + \cdots + 2^{\lfloor \log_2 n \rfloor} = \frac{2^{\lfloor \log_2 n \rfloor + 1}}{2-1} = 2n - 1 = O(n)$.

(v) Let the length of the input x be m, where $m \leq L = \lfloor \log_2 n \rfloor$. There may be at most L distinct variables. The direct test for the satisfiability of a formula of L variables takes $2^L \times poly(L)$ steps. The number of inputs are $O(n)$. So the total time is $O(n \times n \times poly(\log n)) = o(n^3)$.

(vi) The cost of the recursive calls is $\leq 2nT(L)$. We assume that the values of $H()$ for every input of length $\leq L$ are stored and are not recomputed for each of K machines.

(vii) Let $T(n)$ be the worst-case running time to compute $H(n)$. The recurrence relation is

$$T(n) \leq 2nT(L) + O(n^2 L^c)$$

Mathematical induction is used to prove that $T(n) \leq O(n^3)$. If this holds for the smaller arguments, then $T(n) \leq 2nO(L^3) + O(n^2 L^c) = O(n^3)$. We assume that the value of $H(n)$ once computed is stored. So it is not computed again when different machines run on the same input.

Lemma 4. $SAT_H \in \mathbf{P}$ if and only if $H(n) \leq c$ for all n, where c is a constant. Therefore, if $SAT_H \notin \mathbf{P}$, $\lim_{n\to\infty} H(n) = \infty$.

Proof: $\underline{SAT_H \in \mathbf{P} \Rightarrow H(n) \leq c}$: there is a polynomial time bounded TM M that decides SAT_H within $k \times |x|^k$ steps. There are infinitely many TMs equivalent to M. Let the constant $c > k$ be such that $M \equiv M_c$ (M_c halts within $c|x|^c$ steps). Let $n > 2^{2^c}$, so $c < \log\log n$. By the definition, the value of $H(n) \leq c$. So $H(n)$ is bounded by $O(1)$.

$\underline{H(n) \leq c \Rightarrow SAT_H \in \mathbf{P}}$: $\forall n \in \mathbb{N}_0,\ H(n) \in \{1, 2, \cdots c\}$. So there is at least one $i \in \{1, 2, \cdots, c\}$ such that $H(n) = i$ for infinitely many $n \in \mathbb{N}_0$.
We claim that M_i decides SAT_H.

If M_i does not decide SAT_H in time $i \times n^i$, there is some x for which M_i does not halt after $i|x|^i$ steps or gives wrong result. Then for every $n > 2^{|x|}$ ($|x| < \log n$), $H(n) \neq i$ (M_i fails on x). This contradicts the fact that there are infinitely many n for which $H(n) = i$. QED.

Corollary 5. If $SAT_H \notin \mathbf{P}$, $\lim_{n\to\infty} H(n) = \infty$.

Proof:(*Theorem*) We assume that $\mathbf{P} \neq \mathbf{NP}$.
If $SAT_H \in \mathbf{P}$, then $H(n) \leq c$. But then SAT can be reduced in polynomial time to SAT_H, $\phi \mapsto \phi 01^{|\phi|^c}$. And the polynomial time algorithm of SAT_H can be used to solve SAT in polynomial time, implies that $\mathbf{P} = \mathbf{NP}$ - a contradiction.

If $SAT_H \in \mathbf{NP}$-complete: then there is a polynomial (n^k) time bounded reduction from SAT to SAT_H. Our assumption is $\mathbf{P} \neq \mathbf{NP}$, so $SAT_H \notin \mathbf{P}$ implies that $H(n)$ is not bounded above, and $\lim_{n\to\infty} H(n) = \infty$.

A SAT instance ϕ, $|\phi| = n$, is mapped to a SAT_H instance $\psi 01^{|\psi|^{H(|\psi|)}}$ in time $O(n^k)$, where $\phi \equiv \psi$. For a large n, n^k is smaller than $n^{H(n)}$ as $H(n)$ is not bounded. The length $|\psi| + 1 + |\psi|^{H(|\psi|)} < |\phi| + 1 + |\phi|^{H(|\phi|)}$. We have $|\phi|^k = |\psi| + 1 + |\psi|^{H(|\psi|)} \approx |\psi|^{H(|\psi|)}$. Therefore $|\psi| = |\phi|^{k/H(|\psi|)} = o(n)$. In fact $|\psi|$ should be smaller than $|\phi|$ by a polynomial factor ($H(n)$ is computed in polynomial time) e.g. $|\psi| = \sqrt[3]{|\phi|}$ (?). But then ϕ are ψ are satisfiable by the same sets of assignments. This reduction if applied repeatedly gives a formula whose satisfiability can be tested in $O(1)$ time. But then SAT can be decided in polynomial time - a contradiction. QED.

Exercise 2. Prove that if there is a polynomial time reduction from SAT to itself, such that $\phi \mapsto \psi$ and $|\psi| = \sqrt[c]{\phi}$, $c \geq 3$, then $SAT \in \mathbf{P}$.

IMPORTANT - do not remove **Ans.** First we observe two things.

$$n^{1/\ln n} = (e^{\ln n})^{1/lnn} = e.$$

Let $c \geq 3$ be a constant.

$$c^{\ln \ln n} = (e^{\ln c})^{\ln \ln n} = (e^{\ln \ln n})^{lnc} = (\ln n)^d,\ \ln c = d > 1.$$

$$n^{1/(c^{\ln \ln n})} = n^{1/(c^{\ln \ln n})} = n^{1/(\ln n)^d} < n^{1/\ln n} = e.$$

Let the reduction is bounded by n^c. If we perform the reduction $\log \log n$ times. the number of steps are $O(n^c \ln \ln n)$, then the length of the reduced formula is $n^{1/(c^{\ln \ln n})} < e$. Its satisfiability can be checked in $O(1)$ time.
So the membership of SAT can be tested in polynomial time.

SAT_H is not a very natural or useful language; it is deliberately created. But there are languages such as *factoring* and *graph isomorphism* that can be good candidates for **NP** intermediate languages.
Factoring: given three positive integers a, b, c, decide if a has a prime factor p between b and c.
Graph Isomorphism: given two $n \times n$ adjacency matrices corresponding to two graphs G_1 and G_2, decide whether the graphs are same upto renaming of vertices.
They are known to be in **NP**. The certificate for the first is the p, a prime, and the certificate for the second is the permutation of vertices.
No polynomial time algorithm is known for these two problems, and people strongly believe that they are not **NP**-hard.

1.2 Limits of Diagonalization

Diagonalization is used to prove the existence of undecidable decision problems, to prove hierarchy theorems, and the existence of **NP**-intermediate language. But here we show some limitations of the method in connection to the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question. People believe that even clever diagonalization may not be sufficient to answer this question.

We use a variant of Turing machine known as the *Oracle Turing Machine (OTM)*; which was invented by Turing to define the hierarchy of undecidable problems. An *oracle* is a *black box* that can solve the membership problem of a language $O \subseteq \{0,1\}^*$ without incurring any computation cost (time or space). The original Turing machine model can be modified to associate an *oracle* to it.

It is natural for an OTM to solve more problems with less resource modulo the given oracle. These machines define *relativised complexity classes*.

An *Oracle Turing Machine* M^O, where O is the oracle language, has a different read-write tape called a query or oracle tape and three special states s_q, s_y, s_n. The OTM M^O may write a query string u on the query tape and enter the state s_q. The query is, "does u belong to O?"[3]. If the answer of the oracle is '**yes**', the machine enters the state s_y; otherwise it enters the state s_n. Then the computation continues. The query processing does not consume any resource and is considered a single step of computation. We can define new language classes modulo or relative to the query language. The output of M with the oracle language $O \subseteq \{0,1\}^*$ on the input $x \in \{0,1\}^*$ is $M^O(x)$.

Definition 1. The class $\mathbf{P}^O$ is the class of languages decided by a polynomial-time bounded OTM with an oracle for the language O. The class $\mathbf{NP}^O$ is defined similarly.

Example 2. The language $\overline{SAT}$, a **coNP**-*complete* language, belongs to $\mathbf{P}^{SAT}$. The following is the $\mathbf{P}^{SAT}$ OTM.

D: input x

1. If x is not a proper encoding of a CNF Boolean formula, *reject*.
2. Write x in the oracle tape, and enter the state s_q.
3. If the machine enters in s_n, *accept*, else *reject*.

Example 3. $\mathbf{coNP} \subseteq \mathbf{P}^{SAT}$.

If $L \in \mathbf{coNP}$, then $L \leq_p \overline{SAT}$. So L can be decided by a polynomial time OTM through the reduction with SAT as the oracle.

Example 4. If $O \in \mathbf{P}$, then $\mathbf{P^O} = \mathbf{P}$.

Any techniques of *diagonalization* use the following two facts about Turing machines.

(i) Any TM M can be encoded as a string (in fact, infinitely many strings) and there is a TM $\mathcal{U}$ that can decode the description of M and simulate/apply M on its data.

(ii) The simulation overhead in terms of time and space is not much.

In a proof contradiction using *diagonalisation*, a machine M simulates another machine N, and depending on the outcome of the simulation, behaves differently.

An OTMs with the same oracle O also satisfies the two properties mentioned above. Any result about complexity classes that can be proved only with the consideration of (i) and (ii) of ordinary TMs also holds for OTMs using the same oracle. This is called a *relativizing result*.

If $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question can be answered only on the aforementioned properties of Turing machines and diagonalization, then that proof can be extended to $\mathbf{P^O}$ and $\mathbf{NP}$ with every oracle O. If there is a proof of $\mathbf{P} \neq \mathbf{NP}$, then with every oracle O $\mathbf{P^O} \neq \mathbf{NP^O}$.

But the following two theorems show that the question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ cannot be a relativizing result.

Theorem 6. (*Baker, Gill, Solovay* (1975)) There are oracles A and B such that $\mathbf{P^A} \neq \mathbf{NP^A}$ and $\mathbf{P^B} = \mathbf{NP^B}$

[3]The oracle computes a Boolean-valued function.

Proposition 1. There is a language B such that $\mathbf{P}^B = \mathbf{NP}^B$.

Proof: Whatever be the language B, $\mathbf{P}^B \subseteq \mathbf{NP}^B$. We prove the other direction of the subset relation by taking B as a **PSPACE**-*complete* language[4]. Let $L \in \mathbf{NP}^B$. We claim that $L \in \mathbf{PSPACE}$. Let N^B be an OTM that decides L. As N is a polynomial-time-bounded machine, it can send at most a polynomial number of quaries to the oracle B. If B is decided by a polynomial space bounded machine M_B. Then M_B can process (replacing the oracle B) polynomial number of queries of N^B is polynomial space. The non-query processing part of computation of N, also cannot take more than polynomial space. So the whole computation can be carried out in polynomial space.

$L \in \mathbf{PSPACE}$ and B is **PSPACE**-complete, so $L \leq_p B$. Let f be the polynomial-time computable function used for the reduction. The following is a polynomial-time oracle decider for L using f.

D^B: input x

1. Compute $f(x)$.
2. Ask whether $f(x) \in B$.
3. If the answer is '*yes*', *accept*; else *reject*.

As $f(x)$ can be computed in polynomial time, $L \in \mathbf{P}^B$. QED.

Proposition 2. There is a language A such that $\mathbf{P}^A \neq \mathbf{NP}^A$.

Proof: For any language A we define a language L_A as follows:

$$L_A = \{x \in \{1\}^* : \exists y \in A, \ s.t. \ |x| = |y|\}.$$

A strings of length $n \geq 0$ from $\{1\}^*$ is in L_A if and only if there is a string of length n in A. We claim that $L_A \in \mathbf{NP}^A$.

N^A: input $x \in \{1\}^*$

1. Nondeterministically guess a string y whose length is equal to that of x.
2. Ask the oracle about the membership of y in A.
3. If the answer is '*yes*', *accept* x.

We construct the language A in such a way that L_A is not in $\mathbf{P}^A$.

Construction of A: Let M_i be the OTM encoded by the binary representation of i. We construct the set A in stages. The stage i ensures that M_i^A does not decide L_A within $\frac{2^n}{10}$ steps. The set A is empty at the beginning. At each stage, the membership status of a finite number of strings in A is determined.

Stage i: Upto the stage $(i-1)$, a finite number of strings are known to be in A. Take n larger than the lengths of any string already present in A. Simulate M_i on the input 1^n for $\frac{2^n}{10}$ steps. M_i queries the oracle about the status of strings in A, the answer is consistent if the status has already been decided. If the status has not yet been determined, the answer is 'not in A'.
At the end of computation of M_i on 1^n, we ensure that the answer of M_i on 1^n is *incorrect* i.e. $M_i(1^n) \neq L_A(1^n)$.

[4]We can also take any **EXP**-*complete* language, e.g., L_{exp}.

In $2^n/10$ steps, M_i can send queries only about $\frac{2^n}{10}$ strings of length n, and all queries will have negative answers. Then there are two possibilities.
(i) If M_i^A *accepts* 1^n then we ensure that all remaining n-length strings are also not in A. So $M_i^A(1^n) = 1$ but $1^n \notin L_A$.
(ii) If $M_i^A(1^n) = 0$, a string x of lenght n is picked up for which M_i has not sent any query, and added it to A. So $M_i^A(1^n) = 0$, but $1^n \in L_A$.
So $M_i(1^n) \neq L_A(1^n)$. Any polynomial $p(n)$ is smaller than $\frac{2^n}{10}$, for large n, and every TM can be represented by infinitely many strings (longer and longer). So no polynomial time bounded OTM can decide L_A, i.e., $L_A \notin \mathbf{P}^A$.

QED.

People say that *diagonalization* or any simulation method may be useful only if it uses some *non-relativizing* fact.

References

[MS] *Theory of Computation* by Michael Sipser, (3rd. ed.), Pub. Cengage Learning, 2007, ISBN 978-81-315-2529-6.

[CHP] *Computational Complexity* by Christos H Papadimitriou, Pub. Addision-Wesley, 1994, ISBN 0-201-53082-1.

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.

[LAD75] *On the structure of polynomial time reducibility* by R E Ladner, J. ACM, 22(1): 155-171, 1975.