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

Lecture IV

Instructor: Goutam Biswas *Autumn Semester 2023*

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*.

Let $f(n) \geq n$ be a *time constructible* function. We define a time bounded version of H_{TM} as follows.

$$H_f = \{< M, x >: \text{ the DTM } M \text{ accepts the input } x \text{ in } f(|x|) \text{ steps}\}.$$

It is not difficult to see that H_f is *decidable*[1]. But we have the following theorem.

Theorem 1. $H_f \notin \mathbf{DTIME}(f(\lfloor \frac{n}{2} \rfloor))$

There is no $f(\lfloor \frac{n}{2} \rfloor)$ time bounded DTM that decides H_f. The proof is by reduction to a contradiction using diagonalization.

Proof: Suppose an $f(\lfloor \frac{n}{2} \rfloor)$ time bounded DTM N_f decides H_f i.e. N_f on input $< M, x >$ always halts within $f\left(\left\lfloor \frac{|<M,x>|}{2} \right\rfloor\right)$ steps. If "M accepts x in time $f(|x|)$", N_f halts with 'Y'. But if "M rejects x within time $f(|x|)$" or "M does not halt within $f(|x|)$ steps", then N_f halts with 'N'.

Construct the *diagonalizing* machine D_f as follows:

D_f:
Input: $< M >$
  Simulate N_f on $< M, M >$.
  if N_f halts at 'N', then halt at '**Y**'.
  if N_f halts at 'Y', then halt at '**N**'.

[1] $H_f \in \mathbf{DTIME}(f(n)^3)$

The running time of D_f on input $< M >$ ($|M| = n$) is the running time of N_f on input $< M, M >$. So it is $\leq f\left(\left\lfloor \frac{|<M,M>}{2} \right\rfloor\right) = f\left(\lfloor \frac{2n+1}{2} \rfloor\right) = f(n)$ i.e. D_f always halts within $f(n)$ time.

We apply D_f on its own description $< D_f >$. D_f will halt within $f(|D_f|)$ number of steps. There are two possibilities.

If $D_f(< D_f >) =$ 'Y', then $N_f(< D_f, D_f >) =$ 'N' $\equiv < D_f, D_f > \notin H_f$. As D_f always halts within $f(n)$ time, $< D_f, D_f > \notin H_f \Rightarrow D_f(D_f) =$ 'N'.

If $D_f(< D_f >) =$ 'N', then $N_f(< D_f, D_f >) =$ 'Y' $\equiv < D_f, D_f > \in H_f \equiv D_f(< D_f >) =$ 'Y'. Our assumption leads to contradiction. QED.

Theorem 2. (*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

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

Note the following facts before we go for proof:

- Each $x \in \{0,1\}^*$ represents a TM[2]
- There are infinitely many descriptions of a Turing machine.
- If a TM M halts on an input x within $O(f(n))$ time, then its simulation by an universal TM $\mathcal{U}$ halts within $O(f(n) \log f(n))$ time.

Proof: The proof is for a simpler result,

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

Consider the following DTM D.
D :
Input: x

1. Runs the Universal Turing machine $\mathcal{U}$ for $|x|^{1.4}$ steps to simulate M_x (the DTM encoded by x) on x.
2. If the simulation halts with output '**N**' (*reject*), then halt with output '**Y**' (*accept*). If the simulation halts with output **Y** or does not come to halt within this time, then halt with **N**.

As the DTM D halts within time $O(|x|^{1.4}|)$, $L(D) \subseteq \mathbf{DTIME}(n^{1.5})$.
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 $c|x|$ and $M(x) = D(x)$, where c is a constant.

A Universal Turing machine $\mathcal{U}$ can simulate M on input x in time $c_1 c|x| \log |x|$. 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 $D(x) = Y$ if and only if $M_x(x) = M(x) = D(x) = N$ - a contradiction. QED.

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

Theorem 3. (*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 any language decidable in $o(f(n))$ space.

If M is a decider DTM running in $o(f(n))$ space, then $L = L(D)$ is different from $L(M)$ in at least one element. Here comes the diagonalization. M will be simulated on an extension of the string of M's description.

Note that the *asymptotic* behavior sets in beyond some length of 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 $f(n)$. The reason for *padding* the input is to get more work-tape space for these inputs of M.

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

1. Let $|x| = n$.
2. Compute $f(n)$ (space constructible) and mark its end. Halt with 'N' if computation goes beyond the marked end.
3. If $x \neq\!< M > 01^*$ for some DTM M, halt with 'N'.
4. Simulate M on $x =< M > 01^*$ for at most $2^{f(n)}$ steps. If it goes beyond this, M will not halt on x, halt with 'N'. The number of configurations cannot be more than $2^{f(n)}$ with $f(n)$ space[3].
5. If M halts on 'Y', then halt on 'N'; otherwise halt on 'Y'.

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 $x =< M > 01^{n_0}$, the simulation of M on $x =< M > 01^{n_0}$ will be over in stage (4.). 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 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. People asked whether problems in **NP** are of two categories, they are either in **P** or **NP**-complete. The answer is not known but the following theorem in [LAD75] says something very interesting about **NP** intermediate languages.

Theorem 4. If $\mathbf{P} \neq \mathbf{NP}$, then there is a language $L \in \mathbf{NP} \setminus \mathbf{P}$ that is not **NP**-complete.

[3]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.

The language SAT_H is defined as follows.

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

The function $H : \mathbb{N}_0 \to \mathbb{N}_0$ is defined below. But before that we have the following observation. If $H(n) = c$, a constant for all n, then SAT_H is **NP**-complete. It is so because SAT can be reduced to SAT_H in polynomial time, $\phi \mapsto \phi 01^{n^c}$, copy ϕ and pad it with n^c 1's separated by 0. But if $H(n) = n$, then $SAT_H \in \mathbf{P}$. The reason is 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 of the length of $\phi 01^{n^n}$. In fact ϕ cannot be reduced to $\phi 01^{n^n}$ in polynomial time for large $n = |\phi|$.

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 not **NP**-complete and also not in **P** under the assumption $\mathbf{P} \neq \mathbf{NP}$.

Definition 1.

$$H(n) = \begin{cases} i, & i = \min\{j : j \text{ satisfies } C\} \\ \log\log n, & \text{otherwise.} \end{cases}$$

C: j is a natural number less than $\log\log n$, such that the TM M_j decides the membership of x in SAT_H, for all $x \in \{0,1\}^*,\ |x| < \log n$, within $j \times |x|^j$ steps.

A suitable base case is defined.

The computation of $H(n)$ is indirectly recursive as it tests for the membership of SAT_H, which in tern requires the computation of $H(n)$. But then $H(n)$ is well-defined as its value depends on elements of SAT_H whose lengths are within $\log n$. What is the computation time for $H(n)$? Following is an answer.

(i) Is $H(n)$ non-decreasing?

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

(iii) On how many input each machine runs?

(iv) What is the upper bound of time to check the membership of ϕ in SAT for all input?

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

Ans.

(i) The function $H(n)$ is non-decreasing. Let $a < b$, $H(a) = k$ and $H(b) = l$. By the definition the TM M_k correctly predict the membership of all input of length $\log a$, and the TM M_l correctly predict the membership of all input of length $\log b$. As $a < b$, $\log a < \log b$. If $l < k < \log\log a$, the value of $H(a)$ cannot be k ($\min\{j : j \text{ satisfies } C\}$).

(ii) There are at most $\log\log n$ machines to simulate.

(iii) There are $1 + 2 + 2^2 + \cdots + 2^{\lfloor \log n \rfloor}$ inputs as the length of the string is at most $\log n$. So the number of input is $2n - 1$.

(iv) As $H(n)$ is not known, we use the whole input to compute an upper bound of the number of steps to test the membership of $\phi \in$ SAT. If $|x| = k$, the time to construct the corresponding truth table is 2^k. For a length k there are 2^k input strings. Each one takes 2^k steps. So the upper bound of running time of a TM on all input is $\sum_{k=0}^{\lfloor \log n \rfloor} 2^{2k} = \frac{4^{\log n+1}-1}{3} = O(n^2)$.

(v) Considering $\log \log n$ number of TMs, the running time is $n^2 \log \log n = O(n^3)$.

Lemma 5. $SAT_H \in \mathbf{P}$ 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: $SAT_H \in \mathbf{P} \Rightarrow H(n) \leq c$: 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 natural number $c > k$ be such that $M \equiv M_c$. Let $n > 2^{2^c}$, so $c < \log \log n$. By the definition of $H(n)$, its value is $\leq c$. Moreover $H(n)$ is non-decreasing. So $H(n)$ is bounded by c.

$H(n) \leq c \Rightarrow SAT_H \in \mathbf{P}$: $\forall n \in \mathbb{N}_0$, $H(n) \in \{1, 2, \cdots c\}$. So there is an $i \in \{1, 2, \cdots, c\}$ such that $H(n) = i$ for infinitely many $n \in \mathbb{N}_0$. We know that M_i is a polynomial time bounded TM that decides membership of $x \in \{0,1\}^*$, $|x| \leq \log n$ in SAT_H. We claim that M_i decides SAT_H.
Suppose M_i does not give correct membership result for some input x. Let $n > 2^{|x|}$ i.e. $|x| < \log n$. So by the definition $H(m) \neq i$ for all $m \geq n$ - a contradiction to the fact that there are infinitely many n for which $H(n) = i$.

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 each $n \in \mathbb{N}_0$ and $n > 2^{|x|}$. As $|x| < \log n$ and M_i fails on x, implies $H(n) \neq i$. This contradicts the fact that there are infinitely many n for which $H(n) = i$. QED.

Corollary 6. 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 $SAT_H \in$ **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 $\psi 01^{|\psi|^{H(|\psi|)}}$. For a large n, n^k is smaller than $n^{H(n)}$ as $H(n)$ is not bounded. So the size of the SAT_H instance is smaller than $|\phi|^{H(|\phi|)}$ (due to polynomial time reduction). We have $|\phi|^k = |\psi| + |\psi|^{H(|\psi|)} \approx |\psi|^{H(|\psi|)}$. Therefore $|\psi| = |\phi|^{k/H(|\psi|)} = o(n)$. In fact $|\psi|$ should be smaller than $|\phi|$ by a polynomial factor (after all $H(n)$ 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 $SAT \in \mathbf{P}$ - a contradiction. QED.

The language SAT_H is not very natural or useful, it is deliberately created. But there are languages such as *factoring* and *graph isomorphism* who can be good candidate for **NP** intermediate language.
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.
Example 1.

(i) What is the value of $n^{1/\ln n}$?

(ii) Show that $n^{\left(\frac{1}{c}\right)^{\log\log n}}$ is less than a constant, where $c \geq 3$ is a constant.

Ans.

(i) Let
$$n^{1/\ln n} = x \Rightarrow \ln x = \frac{1}{\ln n} \cdot \ln n = 1 \Rightarrow x = e.$$

(ii) $c^{\ln\ln n} = \left(e^{\ln c}\right)^{\ln\ln n} = (e^{\ln\ln n})^{\ln c} = (\ln n)^d$, where $d = \ln c > 1$.
Therefore
$$n^{\left(\frac{1}{c}\right)^{\ln\ln n}} = n^{\frac{1}{(\ln n)^d}} < n^{\frac{1}{\ln n}} < e.$$

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

Ans. Let the reduction be bounded by n^c steps. If we perform the reduction $\log\log n$ times. The number of steps are $O(n^c \log\log n)$. The length of the reduced formula is $n^{\left(\frac{1}{c}\right)^{\ln\ln n}}$, less than a constant. Its satisfiability can be checked by a constant size table look-up in $O(1)$. So the membership of ϕ can be tested in polynomial time through the reduction.

1.2 Limit of Diagonalization

An *oracle* is a *device* that can solve the membership problem of a language without any computation cost (time or space). The original Turing machine model can be modified by supplying an *oracle* to it. It is natural for an *Oracle Turing Machine* (OTM) to solve more problems with less resource modulo the given oracle. This defines *relativised complexity classes*.

An *oracle Turing Machine (OTM)* M^O, where O is the oracle language, has a different read-write tape called a query 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 enters the state s_q. The query is, "does u belong to O?". If the answer from the oracle is '**yes**', the machine is in state s_y and continues. It is i in s_n if the answer is '**no**'. The query processing does not consume any resource.

We can define new language classes modulo or relative to the query language.

Definition 2. The class $\mathbf{P}^O$ is the class of languages decided by a polynomial-time bounded OTM with the oracle of language O. The class $\mathbf{NP}^O$ is defined similarly.

Example 3. The language $\overline{SAT}$, a complete problem of **coNP** belongs to $\mathbf{P}^{SAT}$.

D: input x

1. If x is not a proper encoding of a CNF Boolean formula, *reject*.
2. Ask whether $x \in SAT$.
3. If not, *accept*, else *reject*.

Finally we conclude that through Karp reduction every element of **coNP** belongs to $\mathbf{P}^{SAT}$.

The technique of *diagonalization* uses essentially following two facts about TMs.

(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 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 L 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 TM, also holds for OTMs using same oracle. This is called *relativizing result*.

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

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 is in **PSPACE**. Let OTM N^B decides L. As N is a polynomial time bounded machine, it can send at most polynomial many 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 reduction. Following is a polynomial time oracle decider for L using f.

D: input x

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

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

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: 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 $\mathbf{P}^A$. Let $M_1, M_2, \cdots$ be an enumeration of all polynomial time OTMs. We assume that the running time of M_i is n^i ((?)for simplicity). Initially A is assumed to be empty. The construction of A is done in stages. At the i^{th} stage, a finite subset of A is 'decided' by M_i^A. So M_i^A cannot decide L_A. Before the i^{th} stage, the membership status of a finite number of strings of $\{1\}^*$ are known (whether they are in A).

Stage i: Up to the $(i-1)^{th}$ stage we have included (and excluded) a finite number of strings in A. Choose an n greater than the lengths of all strings whose membership in A has already been decided and also $2^n > n^i$, where n^i is the running time of M_i. We extend the set A so that M_i^A accepts 1^n if and only if $1^n \notin L_A$.

The execution of the oracle Turing machine M_i^A is simulated on the input 1^n. During the run M_i^A sends queries to the oracle of A (which is partially constructed). The simulator processes the queries as follows: If the membership status of the query string y is already known ($y \in A$ or $y \notin A$), the answer is given consistently. If the status of y is *unknown*, the answer (from the simulater to M_i^A) is uniformly '*no*' i.e. $y \notin A$. Simulation continues until M_i^A halts after n^i steps either *accepting* or *rejecting* 1^n.

By the defination $1^n \in L_A$ if and only if there is a string of length n in A. If the oracle machine M_i^A wants to accept L_A, either it must detect a string of length n in A and accept 1^n, or it gets negative answers to all query for all strings of length n (2^n in number). But the n^i step bounded machine cannot send query for all $2^n (> n^i)$ strings of length n. So M_i^A halts and takes decision on its input 1^n on *insufficient information*.

If M_i *accepts* 1^n, no strings of length n is included in A so that 1^n is not in L_A. If M_i *rejects* 1^n, we find a string of length n for which M_i has not sent a query (n^i step bounded OTM cannot send query for all 2^n strings of length n), and put it in A so that 1^n is in L_A. Either way, the set A is '*constructed*' one step further so that M_i^A accepts 1^n if and only if $1^n \notin L_A$.

After this the simulation proceeds to the stage $i+1$. Finally no polynomial time OTM can decide L_A. QED.

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.