Decidable and undecidable problems

We have seen that whatever was not computable by FA or PDA was computable by using TM. This gives rise to an interesting question that **whether or not everything is computable?** We will see that the answer is no. In fact, we will even see that "most" problems are not solvable by Turing machines and, therefore, not solvable by computers.
Any language that exist to a corresponding problem can be classified into three:

1. Recursive language.
2. Recursively enumerable language.
3. Non-recursively enumerable language.

We shall consider these languages in the context of TM.

Recursive language:
We call a language L as recursive if $L = L(M)$ for some Turing machine M such that

1. If w is in L, then M accepts and therefore halts.
2. If w is not in L, then M eventually halts, although it never enters an accepting state.

A TM of this type corresponds to our notion of an algorithm, a well defined sequence of steps that always terminates and produces an answer. If we think of the language L as a problem, as will be the case frequently, then problem L is called decidable if it is a recursive language, and it is called undecidable if it is not a recursive language.

Recursively enumerable language:
A language L is said to be recursively enumerable if there is a TM M that accepts L. The TM M' halts on 'YES instances' but for 'NO instances' it may halt in a non-final state or may get into a loop and never halt.

Recursively enumerable languages are sometimes referred to as Turing-acceptable, and recursive languages are sometimes called Turing-decidable, or simply decidable.

Non-recursively enumerable language:
There doesn't exist a TM even for 'YES instances'.

When there does not doesn't exist a TM for a language means that the set of problems in that language is unsolvable. This rise to a natural question that "Whether the set of unsolvable problems are countable or uncountable?"

Claim 1 *The set of all problems are uncountable.*

Proof. We show that the problems that lists binary string is uncountable. It suffices to show that the set of all binary strings is uncountable. Suppose that the set of all binary strings are enumerable in a table T. Let the i^{th} row contains the string α_i.
We will construct a binary sequence not in this table. Let the sequence $\beta = a_1a_2a_3\ldots$, where i^{th} bit of β can be defined as; $a_i = 0$, if $(\alpha_i)_i = 1$ and $a_i = 1$, if $(\alpha_i)_i = 0$.
We can observe that β is not in T, at least one bit is different from any α_i in T. Hence, we contradict the assumption that the table contains all possible infinite binary sequences. This proves that the set of all binary sequences is uncountable. Therefore, the set of all problems are uncountable. □

Claim 2 *The set of all solvable problems are countably infinite.*

Proof. Let the set of solvable be $P = \{P_1, P_2, \ldots\}$. For each problem in P, each string accepted is subset of Σ^*. We know that $\Sigma = \{0, 1\}$, and Σ^* has a standard ordering. Hence, there exists a bijection between Σ^* and $\mathbb{N}$.
Thus Σ^* is countably infinite. Hence, the set of all solvable problems is a strict subset of Σ^*, and cardinality of set of solvable problems is countably infinite. □

Since cardinality of set of solvable problems is countably infinite, the cardinality of set of recursive language is countably infinite.

Theorem 3. *The set of all unsolvable problems are uncountable.*

Proof. The proof is by contradiction. Suppose that the set of all unsolvable problems is countably infinite. We know that the set of all problems is uncountable by Claim 1, and set of all solvable problems is countably infinite by Claim 2. Since set of all problems is countably infinite and set of all solvable problems is countably infinite, the set of all problems is uncountably infinite (the set of all problems is the union of solvable and unsolvable problems), is a contradiction. Hence, the set unsolvable problems are uncountable. □

1 Properties of languages

We know that the language accepted by TM is called Recursively Enumerable (RE) language. A TM accepts a string by going to a final state and halting. It can reject a string by halting in a non-final state or getting into a loop. A TM when started on an input may get into a loop and never halt.

Theorem 4. *Let L_1 be any recursive language. The complement of L_1 is recursive.*

Proof. Let L_1 be a recursive language accepted by a TM M which halts on all inputs as represented in Figure 1.

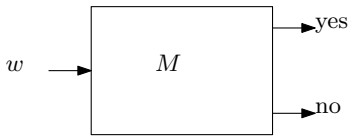

Fig. 1: A recursive TM for L_1

Consider a recursive TM $\overline{M_1}$ as represented in Figure 2, which says yes, when M outputs no. Similarly, $\overline{M_1}$ says no, when M says yes.

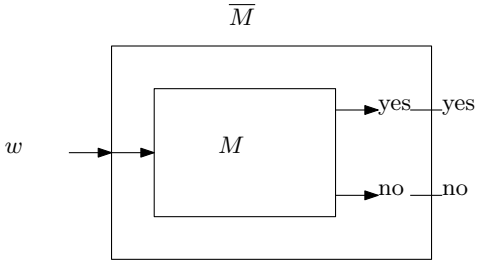

Fig. 2: A recursive TM for $\overline{L_1}$

Therefore, we obtain a recursive TM for $\overline{L_1}$ which halts on all inputs and $\overline{L_1}$ is recursive. □

Consider the language L_2 which is recursively enumerable, then we know that there exists a Turing machine for L_2 which need not always halt. Now, $\overline{L_2}$ can be recursively enumerable or non-recursively enumerable. Suppose that $\overline{L_2}$ is recursively enumerable, then we have the following result:

Theorem 5. *If L_2 be any recursively enumerable language whose complement is also recursively enumerable, then L_2 is recursive.*

Proof. Let L_2 be accepted by M_2, and let $\overline{L_2}$ be accepted by $\overline{L_2}$. Given w, w will either be accepted by M or $\overline{M_1}$. Then L_2 can be accepted by a TM M' which halts on all inputs as represented in Figure 3

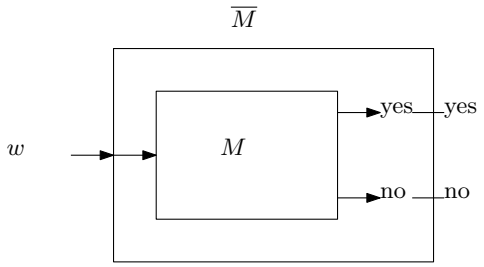

Fig. 3: A recursive TM for L_2

Hence, when L_2 be any recursively enumerable language whose complement is also recursively enumerable, then L_2 is recursive. □

We know that $\overline{L_d}$ is recursively enumerable. Suppose that L_d is recursively enumerable, then we arrive at a contradiction as follows;

Theorem 6. *$\overline{L_d}$ is recursively enumerable but not recursive.*

Proof. We know that $\overline{L_d}$ is not recursive. Suppose that $\overline{L_d}$ is recursive, then the complement L_d will be recursive. But we know that L_d is non-recursively enumerable. Hence $\overline{L_d}$ is not recursive.
Suppose that L_d is recursively enumerable, then we know that $\overline{L_d}$ is recursively enumerable. Hence, L_d is recursive, is a contradiction. Therefore, $\overline{L_d}$ is recursively enumerable and L_d is non-recursively enumerable. □

2 Decidability

The set of languages accepted by DFA or NFA are also accepted by TM. TM with no left move and no write is equivalent to FA. Similarly, the set of languages accepted by DPDA or NPDA are accepted by TM. TM with no left move, no write and write after $ (which is equivalent to stack or push down store).

This shows that a string is either accepted or rejected by FA/PDA. Therefore, the class of languages accepted by FA/PDA is recursive.

We shall look into some questions before we proceed further:

1. **Input:** A TM M, $w \in \Sigma^*$.
   Question: Does M accept w.
2. **Input:** A TM M, $w \in \Sigma^*$.
   Question: Does M stops and halts on w.
3. **Input:** A TM M, w = binary file of M.
   Question: Does M stops and halts on w.
4. **Input:** A TM M.
   Question: Is $L(M)$ regular.
5. **Input:** A TM M.
   Question: Is $L(M)$ context-free.
6. **Input:** TM M_1, TM M_2.
   Question: Is $L(M_1) = L(M_2)$.
7. **Input:** TM M_1, TM M_2.
   Question: Is $L(M_1)OL(M_2)$, where O is some set-theoretic operation.

Can we identify the decidable and undecidable problems in the above list?

3 Undecidability

As we know there is a specific problem that is algorithmically unsolvable. The theorem present here demonstrates that computers are limited in a fundamental way.

3.1 Universal Turing Machines

The Turing machines we have seen so far have been capable of executing a single algorithm.
Before we define universal Turing machine, we shall consider that all Turing machines have same input representation (same input encoding).
A **universal Turing machine** is a Turing machine T_u that works as follows: It is assumed to receive an input string of the form $\langle M, w\rangle$, where M is an arbitrary TM, w is a string over the input alphabet of M. The computation performed by T_u on this input string satisfies these two properties:

1. T_u accepts the string $\langle M, w\rangle$ if and only if M accepts w.
2. If T accepts w and produces output y, then T_u produces output y.

This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine. The universal Turing machine played an important early role in the development of stored-program computers. We can view universal Turing machine as Operating System (OS). Because OS takes input as another programs such as ROM, gcc, etc.,
Remarks:

1. The machine T_u loops on input $\langle M, w\rangle$ if M loops on w, which is why this machine does not decide T_u.
2. The input of T_u can be of any form. For example, $\langle M, w\rangle$, $\langle M_1, M_2, w_1, w_2\rangle$, ...

3.2 The diagnolization method for proving the existence of undecidability

Recall that the strings in Σ^* is countably infinite and we have seen that strings in Σ^* has a standard ordering. Each TM with input alphabet $\{0, 1\}$ can be thought of as a binary string, and we can give mapping to natural numbers. Hence TM is countably infinite and has a standard ordering. Therefore we can talk about i^{th} TM and j^{th} string over $\{0, 1\}$. Consider an infinite Boolean matrix as represented in Figure 4.

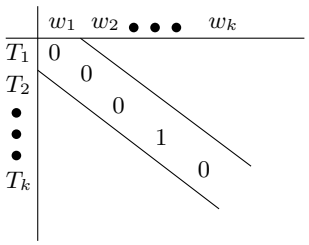

Fig. 4: An infinite Boolean matrix

The entry on the j^{th} row and i^{th} column is 1 if w_i is accepted by T_j and 0 if w_i is not accepted by T_j. Consider the diagonal elements of this infinite Boolean matrix. Take those w_i which correspond to 0 elements in the matrix. We shall define a diagonal language D:

$$D = \{w_i \mid (T_i, w_i) = 0\}$$

Now we ask that **Does there exists a Turing machine to accept or reject D?**

Theorem 7. *D is non-recursively enumerable language.*

Proof. We shall prove by contradiction. Suppose that D is recusively enumerable language. Then, there exists a TM T_k accepting it. If w_k is accepted by T_j, then by definition of D, $w_k \notin D$. But since T_k accepts D, $w_k \in D$. This is a contradiction. Therefore, D is non-recursively enumerable language. □

3.3 The halting problem and the accept problem

Let us consider two decision problems: the general membership problem for recursively enumerable languages, and a problem usually referred to as the halting problem.
Halts: Given a TM T and a string w, does T halt on input w? (The halting problem)

Accepts: Given a TM T and a string w, is $w \in L(T)$? (The accept problem)
In both cases, instances of the problem are ordered pairs (T, w), where T is a TM and w is a string. We shall show that both are undecidable.

Theorem 8. *The halting problem for Turing machine is undecidable.*

Proof. The proof is by contradiction. Suppose the halting problem is decidable. Then, there is an algorithm or there is a TM say T_H that halts on all inputs. We now use T_H as a black box to decide L_d.

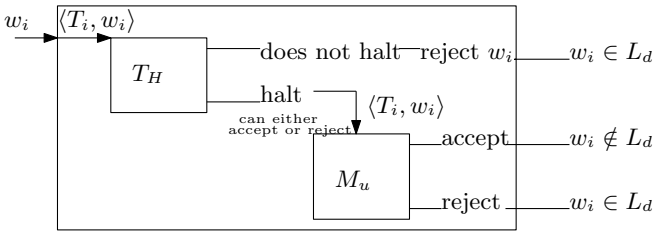

Fig. 5: decidability of L_d

For a string w_i, we identify the corresponding Turing machine T_i using infinite Boolean matrix in Figure 4. We give binary encoding of T_i and w_i to T_H. Since T_H halts on all inputs, it tells true, if T_i halts or false, if T_i doesn't halt on w_i. When T_H says false, reject w_i. Hence $w_i \in L_d$. When T_H says true, T_i can either accept or reject w_i. When T_H halts, using universal Turing machine M_u with input $\langle T_i, w_i \rangle$ we can decide whether $w_i \in L_d$ or $w_i \notin L_d$. When M_u accepts w_i, $w_i \notin L_d$. When M_u rejects w_i, $w_i \in L_d$. This implies that there exists an algorithm for L_d. Hence, L_d is recursive or decidable. This is a contradiction. Therefore, there doesn't exist Turing machine and halting problem is undecidable. □

Consider P-Q problem; recall that P takes Q as input and says 1, if Q runs into an infinite loop, 0, otherwise.

Theorem 9. *Problem P-Q problem is undecidable.*

Proof. Suppose that P-Q problem is decidable. This implies that there exists an algorithm. Thus whether Q halts or not is decidable. Thus Halting problem is decidable, is a contradiction. Thus P-Q problem is undecidable. □

Theorem 10. *The accept problem for Turing machine is undecidable.*

Proof. Suppose that the accept problem is decidable, then there exists a recursive TM T_P which halts for any string w. This implies that the halting problem is decidable, is a contradiction. Hence, T_p cannot exist, and the accept problem is undecidable. □

4 Reducibility between combinatorial problems

Let L and M be languages. Then $L \leq M$ (L is reducible to M) if there is a function f such that $f(x) \in M$ if and only if $x \in L$.

We shall look into the sorting problem which can be solved by using find-min as black box. The sorting problem can be solved using problem of finding minimum in an array, and by making n repeated calls to find-min. We get, first min, second min, and so on upto n^{th} min, which is the solution of the sorting problem. Similarly, 'sorting problem' can be solved using 'in-order traversal' as a black box. The input array is converted into a binary search tree which is in turn passed as an input to in-order traversal algorithm to get a sorting sequence.
Remarks:

1. If find-min is easy, then sorting is easy.
2. If inorder traversal is easy, then sorting is easy.

4.1 Some insights into the reducibility between combinatorial problems

1. If L is easy, then M is easy. ($A \Rightarrow B$)
   If M is hard / difficult, then L is hard/ difficult. (Since we know that $A \Rightarrow B$ if and only if $\neg B \Rightarrow \neg A$)
2. If L is decidable/ solvable/ there exists a algorithm, then M is decidable/ solvable/ there exists a algorithm.
   If M is undecidable/ unsolvable/ there does not exist algorithm, then L is undecidable/ unsolvable/ there does not exist algorithm.

An overview about computability and complexity

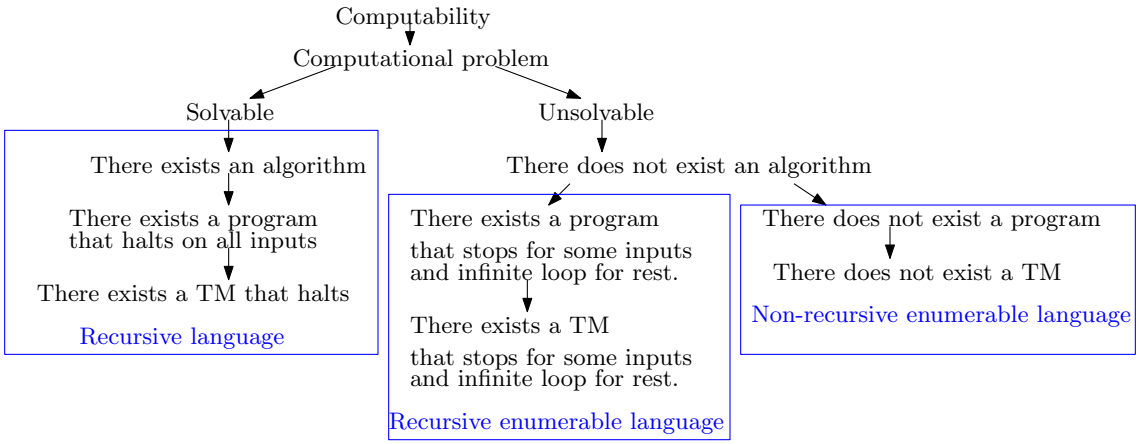

Fig. 6: Computability

If a computational problem is known to be solvable, then the time complexity of the same is analyzed. The time complexity is measured in terms of number of primitive operations. The primitive operation in TM is read/write head movements. Suppose that we analyze the time complexity of $a + b$; To get $a + b$, on reading the input while moving right, when the control sees 0 it changes it to 1. When the control sees \$, it moves left and changes the right most 1 to 0, and halts the machine.
Hence, time complexity of $a + b$ is $a + b + 2 = n + 2$ (which is number of read and write head moves). Thus $n \leq n + 2 \leq 2n$, $T(n) = \theta(n)$. This implies that $a + b$ algorithm is linear in input size and runs in linear time.