Unit 7 - Undecidability & Intractability

- Ankit Pangeni

DATE

* Computational Complexity

The complexity of the computational problems can be measured by choosing a specific abstract machine as a model of computation and considering how much resource machine of that type require for the sowtion of that problem

Complexity measure is a means of measuring the amount of resource used during a computation. Various types of resource such as space & time are used. When estimating these resources, we are always interested in growth rates than absolute values

A problem is regarded as inherently difficult if sowing it requires a very large amount of resources, whatever the algorithm used for solving it. The time & storage are the main Complexity measures to be used. Other measures like amount of communication, no. of gates in a circuit and the no. of processors are also considered.

* Time and space Complexity of Turing Machine

The time and space complexity of a TM can be measured in terms of number of moves and the number of tape squares required for computing an instance of a problem. The most obvious measure of the size of any instance is the length of input string. The worst case is considered as the max time or space required by any string of that length

DATE

The time & space complexity of a TM is defined as:-

- Let T be a TM, the time complexity of T is the function T_t defined on the natural numbers as: for $n \in N$, $T_t(n)$ is the maximum no. of moves T can make on any input string of length n. If there is an input string x such that $|x| = n$, then T loops forever on input T, $T_t(n)$ is undefined.

- The space complexity of T is the function S_t defined as $S_t(n)$ is the maximum number of the tape squares used by T for any input string of length n. If T is multi-tape, no. of tape squares means maximum of the no. of individual tapes. If for some input of length n, it causes T to loop forever, $S_t(n)$ is undefined.

* Intractability

An algorithm for which the complexity measures $s_t(n)$ increases with n, no more rapidly than a polynomial in n is said to be polynomially bounded & one in which it grows exponentially is said to be exponentially bounded

Intractability is a technique for solving problems which are not solvable in polynomial time. The problems that can be solved within reasonable time & space constraints are called tractable. The problems that can't be solved in polynomial time but requires exponential time algorithm are called intractable or hard problems.

PAGE

DATE

To introduce intractability theory, the class P and class NP of problems solvable in polynomial time by deterministic & non-deterministic TM's are essential. When the space & time required for implementing the steps of particular algorithms are reasonable, we can say that problem is tractable. The problems are intractable if the time required for any algorithm is at least $f(n)$, where f is an exponential function of n.

* Complexity classes

There exists some problems whose solutions are not yet found, the problems are divided into classes known as Complexity classes. A complexity class is a set of problems with related complexity. These classes help to group problems based on how much time and space they require to solve problems and verify the solutions. It is the branch of ToC that deals with the resources required to solve a problem.

"The set of problems that can be solved by an abstract machine M using $O(f(n))$ of resource R, where n is the size of the input." is called complexity class

For ex, the class NP is a set of decision problems that can be solved by a non-deterministic TM in polynomial time, while the class P is the set of decision problems that can be solved by a deterministic TM in polynomial space & time

Classmate

PAGE

DATE

NP-Complete (NP-C or NPC) is a class of problems having two properties.

- It is the set of NP problems.
- It is also in the set of NP-hard problems : Any NP problem can be converted into this one by a transformation of the inputs in polynomial time.

Some properties of NP-Complete problems:-

→ No polynomial time algorithms has been found for any of them
→ If polynomial time algorithm is found for any of them, there will be polynomial time algorithm for all of them.
→ If it can be proved that no polynomial time algorithm exists for any of them, then it won't exist for every one of them.
→ It is not the case that there exist no polynomial time algorithm for them, if it is possible, then people can.

* Problems and it's types

i) Abstract Problems :- Abstract problem A is binary relation on set I of problem instances, and the set S of problem solutions. Ex: Minimum spanning tree of a graph G can be viewed as a pair of given graph G & its MST graph T.

ii) Decision Problems :- Decision problem D is a problem that has an answer as either "true", "yes", "1" or "false", "no", "0". Ex: If we have the abstract shortest path with instances of the problem & the solution set as {0,1} then we can transform that abstract problem by reformulating the problem as "Is there a path from u to v with at most k edges". In this situation the answer is yes or no.

PAGE

classmate

DATE

iii) Optimization problems:- We encounter many problems where there are many feasible solutions & our aim is to find the solution with best value. This kind of problem is called optimization problem Ex: Given a graph G, & the vertices u and v, find the shortest path from u to v with minimum number of edges. The NP completeness doesn't directly deal with optimization problems, however we can translate the optimization problem to the decision problem

iv) Function problems: It is a computational problem where a single output is expected for every input, but the output is more complex than that of a decision problem, which isn't just Yes or No. Ex: the travelling salesman problem, which asks for the route taken by the salesman, & the Integer Factorization problem, which asks for the list of factors.

* Reducibility

Reducibility is a way of converging one problem to another in such a way that, a solution to the second problem can be used to solve the first one.

Many complexity classes are defined using the concept of a reduction. A reduction is a transformation of one problem into another problem. There are many different type of reductions based on the method of reduction, such as Cook reductions, Karp & Levin reductions. The most commonly used reduction is polynomial-time reduction. This means the reduction process takes polynomial time.

DATE

Ex, the problem of squaring an integer can be reduced to the problem of multiplying two integers. For complexity classes larger than P, polynomial-time reductions are commonly used

* Turing Reducible

A turing reduction from a problem A to a problem B, is a reduction which solves A, assuming the solution of B is already known. It can be understood as an algorithm that could be used to solve A if it had available to it a subroutine for solving B. More formally, a Turing reduction is a function computable by an oracle machine with an oracle for B. Turing reductions can be applied to both decision & function problems.

* Circuit Satisfiability

The circuit satisfiability problem (also known as CIRCUITSAT, CSAT etc) is the decision problem of determining whether a given Boolean circuit has an assignment of its inputs that makes the output true. In other words, it asks whether the inputs to a given Boolean circuit can be consistently set to 1 or 0 such that the circuit outputs 1. If that is the case, the circuit is called satisfiable. Otherwise, it is unsatisfiable

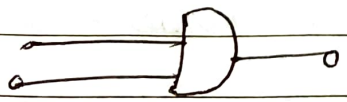


fig: Satisfiable circuit

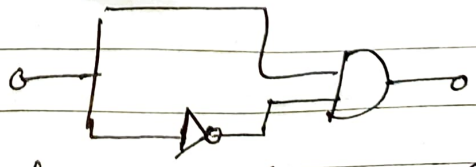


fig: Unsatisfiable circuit

DATE

* Cook's Theorem:-

Lemma: SAT is NP-hard

Proof:-
Take a problem $V \in NP$. Let A be the algorithm that verifies V in polynomial time. We can program A on a computer and therefore there exists a huge logical circuit whose input wires correspond to bits of the inputs x and y of A & which outputs 1 precisely when A(x,y) returns yes.

For any instance x of V. Let Ax be the circuit obtained from A by setting the x-input wire values according to the specific string x. The construction of Ax from x is our reduction function. If x is a yes instance of V, then the certificate y for x gives satisfying assignments for Ax. Conversely, if Ax outputs 1 for some assignments to its input wires, that assignment translates into a certificate for x.

Theorem: SAT is NP-Complete.

Proof: To show that SAT is NP-complete we have to show two properties as given by the definition of NP-complete problems. The first property i.e. SAT is in NP

DATE

Circuit satisfiability problem (SAT) is the question "Given a Boolean combinational circuit, is it satisfiable?" Given the circuit satisfiability problem take a circuit x and a certificate y with the set of vawes that produce output 1, we can verify that whether the given certificate satisfies the circuit in poynomial time. So we can say that circuit satisfiability problem is NP.

This claims that SAT is NP. Now, it is sufficient to show the second property holds for SAT. The proof for the second property ie. SAT is NP-hard is from above lemma. This completes the proof.

Undecidability

An undecidable problem is a decision problem for which it is impossible to construct a single algorithm that always leads to a correct "yes" or "no" answer. It consists of a family of instances for which a particular yes/no answer is required, such that there is no computer program that, for any given problem instance as input, terminates & outputs the required answer after a finite number of steps. More formally, an undecidable problem is a problem whose language isn't a recursive set or computable or decidable. eg:- whether a CFG generates all the strings or not, whether two CFGs L and M are equal? whether the language accepted by TM is regular or CFL? etc

PAGE

classmate

DATE

* Undecidable problems

1. Post's Correspondance Problem (PCP)

It is an undecidable decision problem that was introduced by Emil Post in 1946.

Definition: The input of the problem consists of two infinite lists $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$ of words over some alphabet Σ having at least two symbols. A solution to this problem is a sequence of indices i_k;

$1 \le k \le n$, $\forall k$, such that

$u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$. We say $i_1, i_2, \ldots i_k$ is a solution to this instance of PCP. Here the decision problem is to decide whether solution exists or not.

Ex: Consider the following two lists:

U		
u_1	u_2	u_3
a	ab	bba

V		
v_1	v_2	v_3
baa	aa	bb

A solution to this problem would be the sequence (3, 2, 3, 1) as

$u_3 u_2 u_3 u_1 = bba + ab + bba + a = bbaabbbaa$

$v_3 v_2 v_3 v_1 = bb + aa + bb + baa = bbaabbbaa$

Furthermore, since (3, 2, 3, 1) is a soln, so all of its "repitions" such as (3, 2, 3, 1, 3, 2, 3, 1) etc. are infinitely many soln's of this repititive kind. However, if the two lists had consisted of only u_2, u_3 & v_2, v_3, then there would have been no solution.

PAGE

DATE

2. Halting problem & its proof

"Given a Turing Machine M and an input w, does M halts on w?"

Algorithm may contain loops which may be finite or infinite in length. The amount of work done in an algorithm usually depends on data input. Thus the halting problem asks the question. "Given a program & an input to the program, determine if the program will eventually stop when it is given that input." The question is simply whether the given program will ever halt on a particular input.

Trial solution:-

Just run the program with the given input. If the program stops, we know that the program halts. But if the program doesn't stop in the reasonable amount of time, we cannot conclude that it won't stop. Maybe we didn't wait long enough.

The halting problem is famous because it was one of the first problems proven algorithmically undecidable. This means there is no algorithm which can be applied to any arbitary program & input to decide whether the program stops when run with that input.

DATE

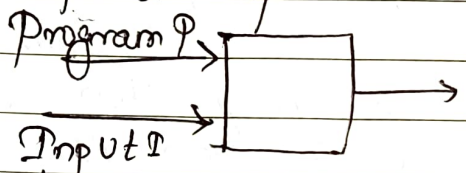
Sketch of a proof that the Halting problem is undecidable.

Suppose, we have a solution to the halting problem called H. Now, H takes two inputs:

i) A program P

ii) An input I for the program P

H generates an output "halt" if H determines that P stops on input I or it outputs "loop" otherwise.

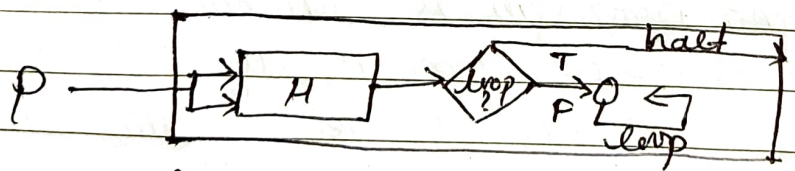


We can treat the program as data & therefore a program can be thought of as input. So, now H can be revised to take P as both inputs (the program & its input) & H should be able to determine if P will halt on P as its input.

Let us construct a new, simple algorithm K that takes H's output as its input & does the following:

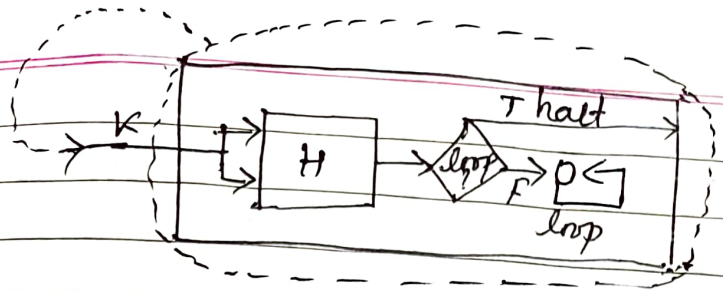
i) If H outputs "loop" then K halts.

ii) Otherwise H's output of "halt" causes K to loop forever.




```
function K(){ if( H()=="loop") { return; }
    else{ while (true); // loop forever } }
```

Since, K is a program, let us use K as the input to K



If H says that k halts then k itself would loop (that's how we constructed it). If H says that k loops then k will halt.

In either case, H gives the wrong answer for k. Thus, H cannot work in all cases. We've shown that it is possible to construct an input that causes any solution H to fail. Hence proved!

* Undecidable problems about Turing Machines:

A problem is undecidable if there is no TM which will always halt in finite amount of time to give answer as 'yes' or 'no'. An undecidable problem has no algorithm to determine the answer for a given input.

Some undecidable problems about TM are:

- Whether two CFGs L & M are equal: Since, we can't determine all the strings of any CFG, we can't predict that two CFGs are equal or not.

- Given a CFL, there is no TM that will always halt an infinite amount of time & gives an answer whether language is ambiguous or not.

DATE

- Completeness of CFG : Given a CFG & input alphabet, whether CFG will generate all possible strings of input alphabet is undecidable
- Regularity of CFL : Given a CFL, determining whether the language is regular is undecidable

Questions asked from this chapter

Q. What do you mean by computational complexity? Explain about the time & space complexity of a Turing Machine. (2078 - 5 marks). (2070 - 5 marks)

Q. Explain the term Intractability. Is SAT problem intractable? Justify. (2078 - 5 marks) (2076 - 5 marks)

Q. What do you mean by tractable & intractable problems? Explain with reference to turing machine. (2076 - 5 marks (2075 - 5 marks)

Q. Differentiate between class P & class NP. (2074 - 5 marks) (2069 - 5 marks, 2071 - 5 marks, 2070 - 5 marks).

Q. What do you mean by problem reduction? Also explain about NP completeness. (2072 - 5 marks)

Q. Show that: If P_1 is NP complete & there is a polynomial time reduction of p_1 to p_2 then p_2 is NP-Complete. (2074 - 5 marks)