Theory of Computation: CS-202

Computability & Complexity

Outline

Computability

- ❑ Primitive Recursive Functions
  - ❑ Initial Functions
  - ❑ Primitive Recursive Functions

Complexity

- ❑ The Classes P and NP
- ❑ Polynomial Time Reduction and NP-completeness
- ❑ Importance of NP-complete Problems
- ❑ SAT is NP-complete
  - ❑ Cook's Theorem

Computability

- The Turing machine is viewed as a mathematical model of a partial recursive function.

- The problem of finding out whether a given problem is 'solvable' by automata reduces to the evaluation of functions on the set of natural numbers or a given alphabet by mechanical means.

- For a given function, if a halting Turing machine exist, we call that function is computable.

Example: $F(x)=2x$

Partial Function

- A partial function f from X to Y is a rule which assigns to every element of X at most one element of Y.

Total Function

- A total function from X to Y is a rule which assigns to every element of X a unique element of Y.

For example.

If R denotes the set of all real numbers, the rule f from R to itself given by f(r) $= +\sqrt{r},$ is a partial function since f(r) is not defined as a real number when r is negative.

But g(r) = 2r is a total function from R to itself.

- A partial or total function f, from X^k to X is also called a function of k variables and denoted by $f(x_1, x_2 \ldots\ldots, x_k)$.

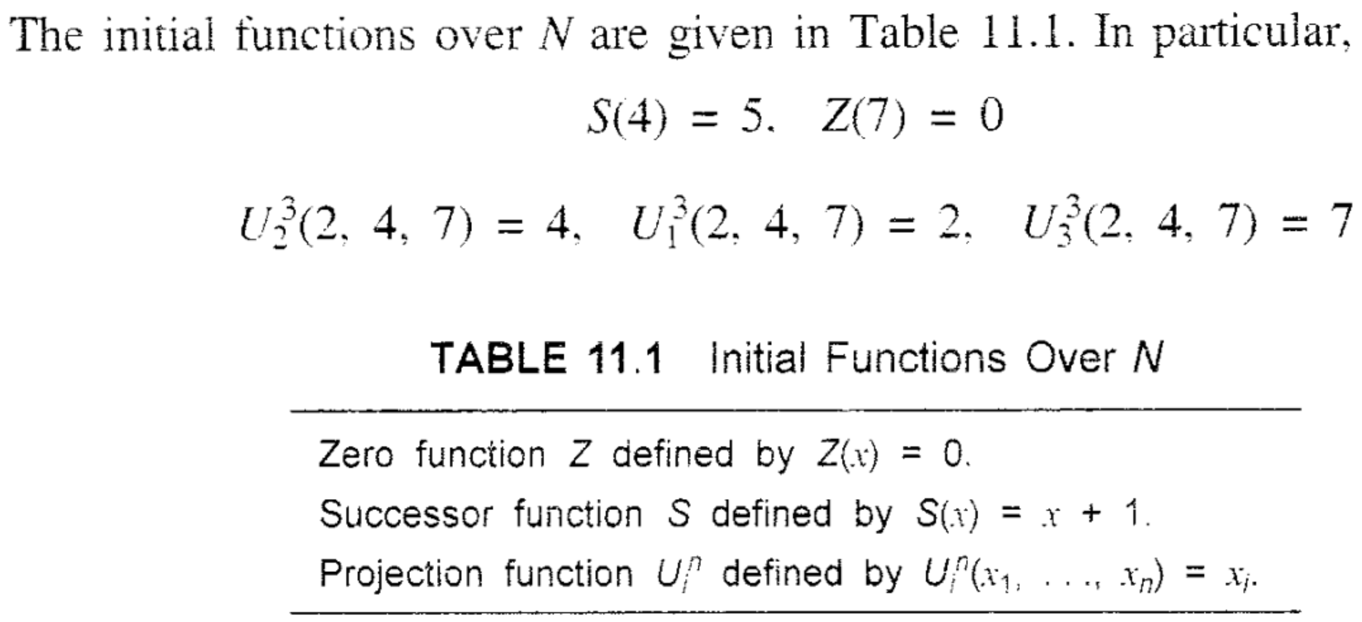
For example. $f(x_1, x_2) = 2x_1 + x_2$ is a function of two variable.

PRIMITIVE RECURSIVE FUNCTIONS

We define some initial functions and declare them as primitive recursive functions.

By applying certain operations on the primitive recursive functions obtained so far. we get the class of primitive recursive.

INITIAL FUNCTIONS



The initial functions over N are given in Table 11.1. In particular,

$$S(4) = 5, \quad Z(7) = 0$$

$$U_2^3(2, 4, 7) = 4, \quad U_1^3(2, 4, 7) = 2, \quad U_3^3(2, 4, 7) = 7$$

TABLE 11.1 Initial Functions Over N

Zero function Z defined by $Z(x) = 0$.
Successor function S defined by $S(x) = x + 1$.
Projection function U_i^n defined by $U_i^n(x_1, \ldots, x_n) = x_i$.

*From Book by K.L.P Mishra

The initial functions over Σ are given in Table 11.2. In particular,

$$\text{nil } (abab) = \Lambda$$

$$\text{cons } a(abab) = aabab$$

$$\text{cons } b(abab) = babab$$

Note: We note that cons $a(x)$ and cons $b(x)$ simply denote the concatenation of the 'constant' string a and x and the concatenation of the constant string b and x.

TABLE 11.2 Initial Functions Over $\{a, b\}$

nil $(x) = \Lambda$
cons $a(x) = ax$
cons $b(x) = bx$

If fl, f2, ..., fk are partial functions of n variables and g is a partial function of k variables, then the composition of g with f1, f2, .. .fk is a partial function of n variables defined by g(f1(xI, x2, …., xn), f2(xI, x2, …., xn), ..., fk((xI, x2, …., xn))

If, for example, f1, f2 and f, are partial functions of two variables and g is a partial function of three variables,
then the composition of g with f1, f2, f3, is given by

g(f1(x1, x2), f2(x1,x2), f3(x1, x2)).

RECURSIVE FUNCTIONS

Let g(x1, x2 ..., xn, y) be a total function over N.

g is a regular function if there exists some natural number Y_0 such that g(x1, x2, .. .,xn, y_0) = 0 for all values x1, x2, ..., xn in N.

For instance, g(x, y) = min(x, y) is a regular function since g(x, 0) = 0 for all x in N.

But f(x, y) = I x-y| is not regular since f(x, y) = 0 only when x = y, and so we cannot find a fixed y such that f(x, y) = 0 for all *x* in *N*.

CONSTRUCTION OF THE TURING MACHINE THAT CAN PERFORM COMPOSITION

- The zero function Z is defined as Zero(a1) = 0 for all a1>=0.
- So the initial tape expression can be taken as X = 1^{a1}xby.
- As we require the computed value Zero(a1) namely 0 to appear to the left of y, we require the machine to halt without changing the input.

- Thus we define a TM by taking Q = {qo, q1}, Γ = {b, x_1, y},
- P consists of q_0bRq_0, q_01Rq_0 , $q_0x_1x_1q_1$ and are used to move to the right until x_1 is encountered.
- $q_0x_1x_1q_1$ enables the TM to enter the state q_1.
- M enters q_1 without altering the tape symbol.
- In terms of change of IDs. we have

$$q_0 1^{a_1} x_1 by \overset{*}{\vdash} 1^{a_1} q_0 x_1 by \vdash 1^{a_1} q_1 x_1 by$$

- As there is no quadruple starting with q_1, M halts

State	b	1	x_1	y
q_0	$(R,\ q_0)$	$(R,\ q_0)$	$(x_1,\ q_1)$	
q_1				

- Construction of the Turing machine that can compute the zero function z.
- Construction of the Turing machine for computing-the successor function
- Construction of the Turing machine for computing the projection $u_i^{m.}$
- Construction of the Turing machine that can perform composition

COMPUTABILITY

- Partial recursive functions introduced in the earlier sections are Turing-computable.
- In mid 1930s. mathematicians and logicians were trying to rigorously define computability and algorithms.
- In 1934 Kurt Gödel pointed out that primitive recursive functions can be computed by a finite procedure (i.e. an algorithm).
- He also hypothesized that any function computable by a finite procedure can be specified by a recursive function.
- Around 1936, Turing and Church independently designed a 'computing machine' (later termed Turing machine) which can carry out a finite procedure.

- For formalizing computability, Turing assumed that while computing, a person writes symbols on a one-dimensional paper (instead of a two- dimensional paper as is usually done) which can be viewed as a tape divided into cells.

- He scans the cells one at a time and usually performs one of the three simple operations.

(i) writing a new symbol in the cell he is scanning,

(ii) moving to the cell left of the present cell,

(iii) moving to the cell right of the present cell.

These observations led Turing to propose a computing machine.

Complexity

Turing Machine Models and Complexity

- When a problem/language is decidable, it simply means that the problem is computationally solvable in principle, It may not be solvable in practice in the sense that it may require enormous amount of computation time and memory.
- The proofs of decidability/undecidability are quite rigorous, since they depend solely on the definition of a Turing machine and rigorous mathematical techniques.
- But the proof and the discussion in complexity theory rests on the assumption that P ≠ NP.
- The computer scientists and mathematicians strongly believe that P ≠ NP. but this is still open.

$P \neq NP$?

- This problem is one of the challenging problems of the 21st century.
- This problem carries a prize money of \$1M.
- P stands for the class of problems that can be solved by a deterministic algorithm (i.e. by a Turing machine that halts) in polynomial time.
- NP stands for the class of problems that can be solved by a nondeterministic algorithm (that is, by a nondeterministic TM) in polynomial time.
- P stands for polynomial and NP stands for nondeterministic polynomial.
- Another important class is the class of NP-complete and NP Hard problems.

GROWTH RATE OF FUNCTIONS

- when we have two algorithms for the same problem, we may require a comparison between the running time of these two algorithms.

- With this in mind. we study the growth rate of functions defined on the set of natural numbers.

An exponential function is a function $q : N \to N$ defined by

$$q(n) = a^n \qquad \text{for some fixed } a > 1.$$

When n increases, each of n, n^2, 2^n increases. But a comparison of these functions for specific values of n will indicate the vast difference between the growth rate of these functions.

Growth Rate of Polynomial and Exponential Functions

n	$f(n) = n^2$	$g(n) = n^2 + 3n + 9$	$q(n) = 2^n$
1	1	13	2
5	25	49	32
10	100	139	1024
50	2500	2659	$(1.13)10^{15}$
100	10000	10309	$(1.27)10^{30}$
1000	1000000	1003009	$(1.07)10^{301}$

THE CLASSE P

P: the class of problems that have polynomial-time deterministic algorithms. – That is, they are solvable in O(p(n)), where p(n) is a polynomial on n.

A deterministic algorithm is (essentially) one that always computes the correct answer

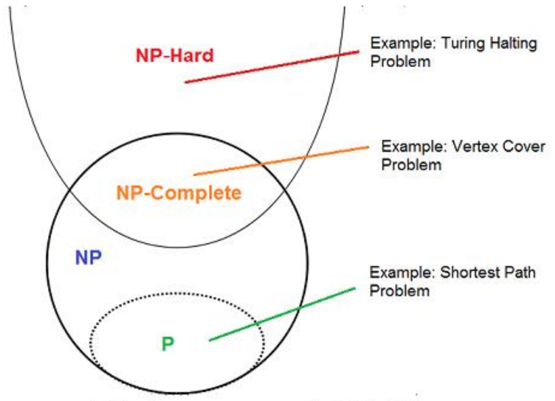
THE CLASS P

A Turing machine M is said to be of time complexity T(n) if the following holds:

Given an input w of length n. M halts after making at most T(n) moves

A language L is in class P if there exists some polynomial T(n) such that L = T(M) for some deterministic TM M of time complexity T(n).

THE CLASS NP


NP-Hard
Example: Turing Halting Problem
Example: Vertex Cover Problem
NP-Complete
NP
Example: Shortest Path Problem
P
This diagram assumes that P != NP

NP: the class of decision problems that are solvable in polynomial time on a nondeterministic machine (or with a nondeterministic algorithm).

– (A deterministic computer is what we know) – A nondeterministic computer is one that can “guess” the right answer or solution.

Thus NP can also be thought of as the class of problems “whose solutions can be verified in polynomial time”.

NP stands for “Nondeterministic Polynomial-time”

Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

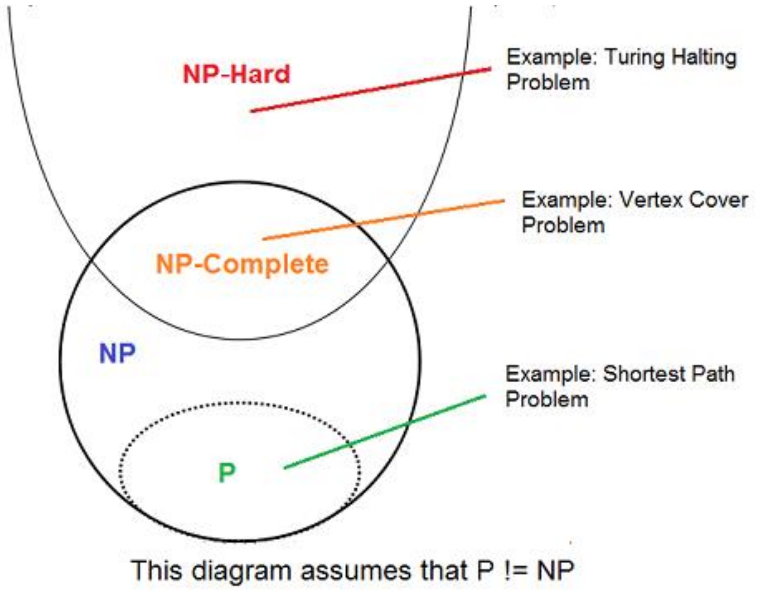
Problem

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Solution

NP-hard

- A problem is NP-hard if all problems in NP are polynomial time reducible to it


NP-Hard
Example: Turing Halting Problem
Example: Vertex Cover Problem
NP-Complete
NP
Example: Shortest Path Problem
P
This diagram assumes that P != NP

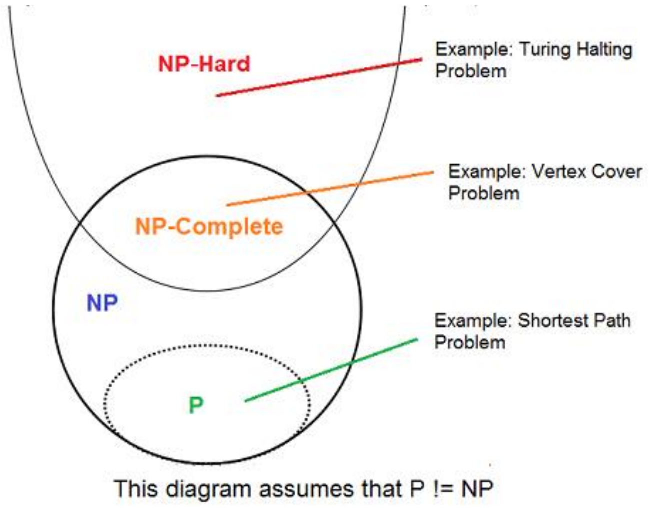
POLYNOMIAL TIME REDUCTION

- Let P_1 and P_2 be two problems. A reduction from P_1 to P_2 is an algorithm which converts an instance of P_1 to an instance of P_2.
- If the time taken by the algorithm is a polynomial p(n), n being the length of the input of P_1. then the reduction is called a polynomial reduction P_1 to P_2.

- If there is a polynomial time reduction from P_1 to P_2 and if P_2 is in P then P_1 is in P.

NP-complete problems

- A problem is NP-complete if the problem is both – NP-hard, and – NP


NP-Hard
Example: Turing Halting Problem
Example: Vertex Cover Problem
NP-Complete
NP
Example: Shortest Path Problem
P
This diagram assumes that P != NP

NP-COMPLETENESS

Let L be a language or problem in NP. Then L is NP-complete if

1. L is in NP.
2. For every language L' in NP there exists a polynomial-time reduction of L' to L.

Note: . The class of NP-complete languages is a subclass of NP.

SAT IS NP-COMPLETE

- we prove that the satisfiability problem for boolean expressions (whether a boolean expression is satisfiable) is NP-complete.

- This is the first problem to be proved NP-complete. Cook proved this theorem in 1971.

COOK'S THEOREM

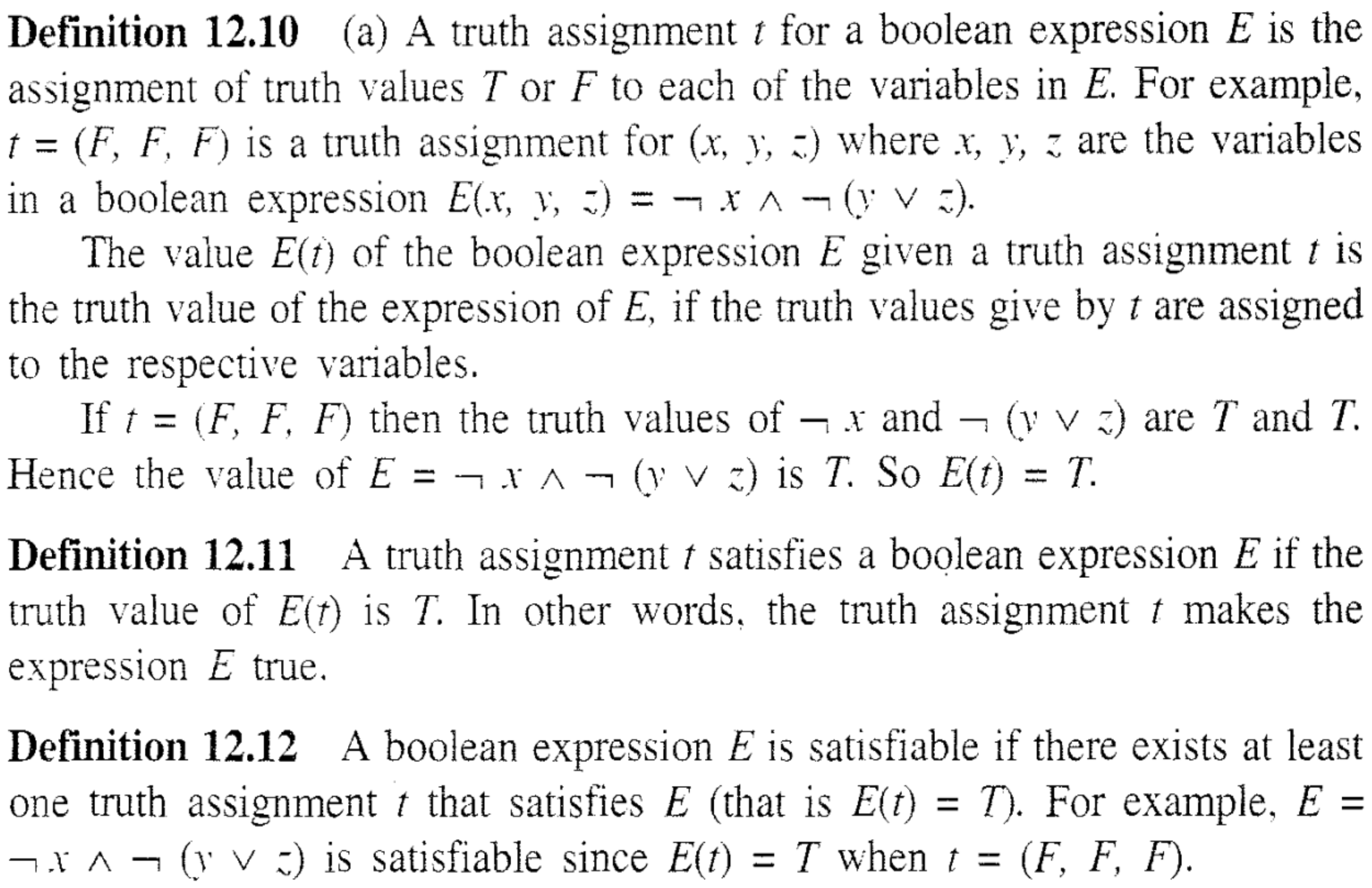
The satisfiability problem (SAT) is the problem:

Given a boolean expression. Is it satisfiable?

Note: The SAT problem can also be formulated as a language.

We can define SAT as the set of all coded Boolean expressions that are satisfiable.

So the problem is to decide whether a given coded boolean expression is in SAT.



Definition 12.10 (a) A truth assignment t for a boolean expression E is the assignment of truth values T or F to each of the variables in E. For example, $t = (F, F, F)$ is a truth assignment for (x, y, z) where x, y, z are the variables in a boolean expression $E(x, y, z) = \neg x \wedge \neg (y \vee z)$.

The value $E(t)$ of the boolean expression E given a truth assignment t is the truth value of the expression of E, if the truth values give by t are assigned to the respective variables.

If $t = (F, F, F)$ then the truth values of $\neg x$ and $\neg (y \vee z)$ are T and T. Hence the value of $E = \neg x \wedge \neg (y \vee z)$ is T. So $E(t) = T$.

Definition 12.11 A truth assignment t satisfies a boolean expression E if the truth value of $E(t)$ is T. In other words, the truth assignment t makes the expression E true.

Definition 12.12 A boolean expression E is satisfiable if there exists at least one truth assignment t that satisfies E (that is $E(t) = T$). For example, $E = \neg x \wedge \neg (y \vee z)$ is satisfiable since $E(t) = T$ when $t = (F, F, F)$.

*From Book K.L.P Mishra

Cover these topics from Book

- Cook’s theorem
- Rice’s theorem
- Church Turing thesis
- Importance of NP-complete problems

Suggested readings

1. An introduction to FORMAL LANGUAGES and AUTOMATA by PETER LINZ.
2. Introduction to Automata Theory, Languages, And Computation by JOHN E. HOPCROFT, RAJEEV MOTWANI, JEFFREY D. ULLMAN
3. Theory of computer science: automata, languages and computation **by** K.L.P MISHRA

Thank you