

Logic and Computation: I

Chapter 2 Propositional logic and computational complexity

Kazuyuki Tanaka

BIMSA

November 24, 2022



北京雁栖湖
应用数学研究院
YANQI LAKE BEIJING INSTITUTE OF
MATHEMATICAL SCIENCES AND APPLICATIONS

Logic and Computation I

- **Part 1. Introduction to Theory of Computation**
- **Part 2. Propositional Logic and Computational Complexity**
- **Part 3. First Order Logic and Decision Problems**

Part 2. Schedule

- Nov.17, (1) Tautologies and proofs
- Nov.22, (2) The completeness theorem of propositional logic
- **Nov.24, (3) SAT and NP-complete problems**
- Nov.29, (4) NP-complete problems about graphs
- Dec. 1, (5) Time-bound and space-bound complexity classes
- Dec. 6, (6) PSPACE-completeness and TQBF

SAT and NP-complete problems

- 1 Recap
- 2 Historical introduction
- 3 The classes P and NP
- 4 Polynomial time reducibility
- 5 NP-complete
- 6 Satisfiability problem
- 7 Cook - Levin theorem
- 8 CNF-SAT
- 9 3-SAT
- 10 Summary

Recap

Historical
introductionThe classes P
and NPPolynomial time
reducibility

NP-complete

Satisfiability
problemCook - Levin
theorem

CNF-SAT

3-SAT

Summary

- Propositional logic is the study of logical connections between propositions.
- If any truth-value function V satisfying all propositions in Γ also satisfies φ , then φ is said to be a tautological consequence of Γ , written as $\Gamma \models \varphi$.
- A sequence of propositions $\varphi_0, \varphi_1, \dots, \varphi_n$ satisfying the following conditions is a proof in Γ : For $k \leq n$,
 - (1) φ_k belongs to $\{P1, P2, P3\} \cup \Gamma$, or
 - (2) There exist $i, j < k$ such that $\varphi_j = \varphi_i \rightarrow \varphi_k$ (MP).

The last component of proof φ_n is called a **theorem** in Γ , written as $\Gamma \vdash \varphi_n$.

- Completeness theorem: $\Gamma \vdash \varphi \Leftrightarrow \Gamma \models \varphi$.
- Completeness theorem (another version): Γ is consistent $\Leftrightarrow \Gamma$ is satisfiable.
- Compactness theorem: If any finite subset of Γ is satisfiable, then Γ is also satisfiable.

Historical introduction (1)

Recap

Historical
introductionThe classes P
and NPPolynomial time
reducibility

NP-complete

Satisfiability
problemCook - Levin
theorem

CNF-SAT

3-SAT

Summary

- **Decision problem** (Entscheidungsproblem) is to determine the existence of a solution or property rather than figuring out a specific solution.
- Gödel's incompleteness theorem (1931) showed that ordinary deductive systems cannot solve the decision problem of arithmetic.
- Turing and Church (1936) solved the decision problem of first-order logic in a completely negative way.
- Turing defined the so-called universal Turing machine. After demonstrating the undecidability of the halting problem K , he expressed the problem as the satisfiability of first-order logic.



K. Gödel



A. Turing

Historical introduction (2)

- For decidable problems, it is important to find efficient algorithms and to show the limits of their efficiency.
- Research on computational efficiency or complexity began in 1965 with the paper “On the Computational Complexity of Algorithms” by J. Hartmanis and R.E. Stearns, published in the Transaction of the American Mathematical Society.

ON THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS

BY

J. HARTMANIS AND R. E. STEARNS

I. Introduction. In his celebrated paper [1], A. M. Turing investigated the computability of sequences (functions) by mechanical procedures and showed that the set of sequences can be partitioned into computable and noncomputable sequences. One finds, however, that some computable sequences are very easy to compute whereas other computable sequences seem to have an inherent complexity that makes them difficult to compute. In this paper, we investigate a scheme of classifying sequences according to how hard they are to compute. This scheme puts a rich structure on the computable sequences and a variety of theorems are established. Furthermore, this scheme can be generalized to classify numbers, functions, or recognition problems according to their computational complexity.



J. Hartmanis



R.E. Stearns

- Two measures of computational efficiency: time and space
- In this lecture, we only consider time complexity, especially polynomial time one.

Polynomial-time solvable, the class P

A decision problem is **polynomial-time solvable** or has a **polynomial-time algorithm** if there exists a deterministic (multi-tape) TM and a polynomial $p(x)$ such that for an input string of length n , it returns the correct answer Yes or No within $p(n)$ steps. The class of all such problems is called **P**.

The class NP

A problem is said to **have a nondeterministic polynomial-time algorithm** if there is a nondeterministic TM and a polynomial $p(x)$ such that for an input string of length n , it always stops within $p(n)$ steps and answers

- ▷ Yes, if at least one accepting computation process admits it;
- ▷ No, if all the computation processes reject.

The class of all such problems is called **NP**.

- More rigorous definitions of classes P and NP will be given in lecture 02-05.
- Note that it is easy (for most of time bounding functions) to make a machine always stop within the time. A TM with a polynomial time bound is called a polynomial-time TM.
- Clearly $P \subset NP$.
- $P = NP$: all NP problems can be solved by polynomial-time algorithms?
- It is widely believed that this is not the case, which is called **$P \neq NP$ conjecture**.

Definition (Polynomial time reducibility)

A problem Q_1 is **polynomial (time) reducible** to Q_2 , denoted as $Q_1 \leq_p Q_2$ if there exists a polynomial-time (deterministic) algorithm A which can solve a problem q_1 in Q_1 as the problem $A(q_1)$ in Q_2 .

- A in the above definition can be seen as a deterministic polynomial-time TM with outputs.
- We solve a problem q_1 in Q_1 as the problem $A(q_1)$ in Q_2 , by taking answer Yes/NO to $A(q_1)$ in Q_2 as answer Yes/NO to q_1 in Q_1 .
- If $Q_1 \leq_p Q_2$, then Q_1 can be solved polynomially by using an algorithm for Q_2 , and so generally Q_2 is more difficult.

Lemma

$$Q_1 \leq_p Q_2 \wedge Q_2 \in P \Rightarrow Q_1 \in P.$$

Definition

- A problem belonging to NP is called a **NP problem**.
- A problem Q (not limited to NP problems) is said to be **NP-hard** if for any NP problem Q' , $Q' \leq_p Q$.
- An NP-hard NP problem is said to be **NP-complete**.

An NP-complete problem is the most difficult problem in NP. If there is a NP-complete problem that can be solved by P, all NP problems can be solved by P, so $P = NP$.

The first NP-complete problem discovered by Cook in 1971 was the satisfiability problem for propositional logic (Boolean algebra). Subsequently, many other NP-complete problems were found by R. Karp.

Definition (Satisfiability problem)

The satisfiability problem **SAT** is to determine whether a give proposition (or a Boolean formula) is satisfiable or not, i.e., whether there exists a truth assignment that makes the proposition (Boolean formula) have value T (or 1).

- From now on, the decision problem Q is identified with the set of (symbol strings representing) individual problems whose answer is Yes. Therefore, SAT is regarded as a set of satisfiable propositions (Boolean formulas).
- Here we use T, F instead of the constants 1, 0 even in Boolean formulas, so that propositions and Boolean formulas are not distinguished. The constants 1, 0 are not used in order to distinguish them from natural numbers.
- For example, for the Boolean formula $\varphi(x_1, x_2) = (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2)$, $\varphi(T, F) = T$, so $\varphi(x_1, x_2) \in \text{SAT}$.

Theorem (Cook - Levin theorem)

SAT is NP-complete.



S. Cook



L. Levin

S. Cook received his PhD from Harvard University under the supervision of mathematical logician Professor Wang Hao. He was an assistant professor in the Department of Mathematics at the University of California, Berkeley at the time of his discovery.

L. Levin, who was Kolmogorov's student in Moscow, independently made a similar discovery. So it is called the Cook-Levin theorem.

Proof.SAT \in NP

- First of all, an input formula must be expressed as a sequence from a finite set of symbols. So, variables x_n 's need some coding. We simply express variables x_0, x_1, x_2, \dots by symbol strings $1, 11, 111, \dots$. And we loosely assume that the variables are used consecutively in order of x_0, x_1, x_2, \dots . Thus, for example, $(\neg x_0 \vee \neg x_1 \vee F) \wedge (x_0 \vee \neg x_2)$ is expressed by $(\neg 1 \vee \neg 11 \vee F) \wedge (1 \vee \neg 111)$. So, a formula with length n can be coded as a symbolic string with length $\leq n^2$.
- Next step is to assign truth values nondeterministically to the variables of a formula φ , and evaluate it. By a sequence TFT, for example, we denote the assignment of T, F, and T to the variables x_0, x_1 , and x_2 , respectively. So, we non-deterministically choose a sequence of T, F by which a given formula is evaluated.

SAT \in NP (continued)

- Now, suppose we choose the sequence TFT and then we evaluate φ as follows.
- In an input string, replace 1, 11, 111 (not part of a longer sequence of 1's) by T, FF, TTT. Then, $(\neg 1 \vee \neg 11 \vee F) \wedge (1 \vee \neg 111)$ becomes

$$(\neg T \vee \neg FF \vee F) \wedge (T \vee \neg TTT).$$

- Repeat the following process. If we find symbol \neg that precedes T or F, replace \neg and all subsequent T's or F's with their opposite truth symbols. Then, the above example becomes

$$(FF \vee TTT \vee F) \wedge (T \vee FFFF)$$

- For subformulas of truth strings connected by \vee or \wedge , replace each symbol with the truth symbol of the result by the operations. Hence we have,

$$(TTTTTTTT) \wedge (TTTTTT)$$

SAT \in NP (continued)

- If a sequence of truth symbols is ended with parentheses, then replace both parentheses with the truth symbols inside.

$$(TTTTTTTT) \wedge (TTTTTT) \implies TTTTTTTTTT \wedge TTTTTTTT$$

- For this example, perform the operation \wedge again, we obtain

TTTTTTTTTTTTTTTTTTTTTTTT

which shows that the assignment TFT satisfies φ . If we obtain a sequence of F's for some assignment, then φ is not satisfied by the assignment.

- The choice of an assignment is non-deterministic. But after that, the calculation is deterministic and must halt in at most n^2 steps for an input string of length n , or at most n^4 steps for a formula of length n . Hence SAT \in NP.

Any NP-problem R can be reduced to SAT in polynomial time

- Let $M = (Q, \Omega, \delta, Q_0, F)$ be a nondeterministic TM to decide (accept) the problem R in polynomial time $p(n)$.
- We construct will a Boolean formula Φ_w representing the computation process of M on input w of length n , and M accepts w in polynomial time $p(n)$ if and only if Φ_w is satisfiable.

Any NP problem R can be reduced to SAT in polynomial time

- In Φ_w , $x_{t,i,a}$ denotes the $i + 1$ th ($\leq n + p(n) + 1$) symbol $a (\in \Omega \cup Q)$ in the computational configuration $a_1 \cdots a_{j-1} q a_j \cdots a_k$ at time $t \leq p(n)$.
- Since t is bounded by $p(n)$, the used area of the tape does not exceed $n + p(n)$. In addition, $\Omega \cup Q$ is a finite set. Therefore, the total number of variables $x_{t,i,a}$ is a constant multiple of $p(n)^2$.
- Suppose M 's input is $w = w_1 \cdots w_n$; other places of the tape are blank with symbol B .
- A Boolean formula φ_w representing the initial configuration ($t = 0$) is defined:

$$\bigvee_{q \in Q_0} \left(x_{0,0,q} \wedge \bigwedge_{a \neq q} \neg x_{0,0,a} \right) \\ \wedge \left(x_{0,1,w_1} \wedge \bigwedge_{a \neq w_1} \neg x_{0,1,a} \right) \wedge \cdots \wedge \left(x_{0,n,w_n} \wedge \bigwedge_{a \neq w_n} \neg x_{0,n,a} \right) \\ \wedge \bigwedge_{i > n} \left(x_{0,i,B} \wedge \bigwedge_{a \neq B} \neg x_{0,i,a} \right). \quad (\text{note: } i \leq n + p(n))$$
- The truth value assignment satisfying φ_w represents the initial config. of M .
- The number of variables included in φ_w is a constant multiple of $p(n)$.

Any NP problem R can be reduced to SAT in polynomial time (continued)

- Next, we consider M 's transition rule $X : (a', L, q) \in \delta(p, a)$

$$\dots bpa \dots \triangleright \dots qba' \dots$$

- We assume that bpa is the $i + 1$ th to $i + 3$ th symbols of computational configuration at time t .
- The relation between $x_{t,--}$ and $x_{t+1,--}$, denoted as $\varphi(t, i, X)$, is expressed as:

$$\begin{aligned} & (x_{t,i,b} \wedge x_{t,i+1,p} \wedge x_{t,i+2,a}) \wedge (x_{t+1,i,q} \wedge x_{t+1,i+1,b} \wedge x_{t+1,i+2,a'}) \\ & \wedge \bigwedge_{c \neq q} \neg x_{t+1,i,c} \wedge \bigwedge_{c \neq b} \neg x_{t+1,i+1,c} \wedge \bigwedge_{c \neq a'} \neg x_{t+1,i+2,c} \\ & \wedge \bigwedge_{j \notin \{i, i+1, i+2\}, a} (x_{t+1,j,a} \leftrightarrow x_{t,j,a}). \end{aligned}$$
- Assuming $\varphi(t, i, X)$ has value \mathbb{T} , the values of $x_{t+1,--}$ are uniquely determined from the truth value assignment of $x_{t,--}$.
- The total number of variables in $\varphi(t, i, X)$ is a constant multiple of $p(n)$.
- Details on other transitions are omitted, since they can also be treated as above.

Any NP problem R can be reduced to SAT in polynomial time (continued)

- Composing the above Boolean formulas, we define:

$$\Phi_w \equiv \varphi_w \wedge \left\{ \bigwedge_{t,i,X} \bigvee \varphi(t,i,X) \right\} \wedge \bigvee_{t,i,q \in F} x_{t,i,q}$$

- The final disjunction means that a final state $q \in F$ appears in the computation.
- Since the variables contained in Φ_w are constant multiples of $p(n)^3$, we construct Φ_w from w by a deterministic polynomial-time algorithm.
- By the definition of Φ_w , it is clear that

$$w \in L(M) \Leftrightarrow \Phi_w \in \text{SAT}$$

- Thus problem R can be reduced to SAT in polynomial time. □

By the completeness theorem, the set of theorems of propositional logic is equivalent to the set of tautologies. So, it turns out to be polynomial-time reducible to the complement of SAT.

$$\vdash \varphi \iff \neg \varphi \notin \text{SAT}.$$

Next we show that SAT for a special class of Boolean formulas remains NP-complete.

Definition

A variable x and its negation $\neg x$ are called **literals**. A disjunction (\vee) of literals is called a **clause**. A conjunction (\wedge) of clauses is called a **CNF** (conjunctive normal form).

Definition

CNF-SAT is a satisfiability problem for conjunctive normal forms.

Theorem

CNF-SAT is NP-complete.

Proof.

- Since $\text{CNF-SAT} \subset \text{SAT}$, $\text{CNF-SAT} \in \mathbf{NP}$ is obvious from $\text{SAT} \in \mathbf{NP}$.
- Goal: $\text{SAT} \leq_p \text{CNF-SAT}$.
- We are looking for a polynomial-time algorithm that converts a Boolean formula to an equivalent CNF. But this is not easy. For instance, if you convert

$$(\varphi_1^1 \wedge \varphi_2^1) \vee (\varphi_1^2 \wedge \varphi_2^2) \vee \cdots \vee (\varphi_1^n \wedge \varphi_2^n)$$

to

$$\bigwedge_{1 \leq i_1, i_2, \dots, i_n \leq 2} (\varphi_{i_1}^1 \vee \varphi_{i_2}^2 \vee \cdots \vee \varphi_{i_n}^n),$$

then the formula size increases from $2n$ to n^2 . So, if one repeats, its size would be beyond polynomial.

Proof(continued)

- However, there is no need to construct an equivalent CNF. It is sufficient to show that for any formula φ , there is a CNF formula ψ such that

$$\varphi \in \text{SAT} \iff \psi \in \text{SAT}. \quad (1)$$

In fact, the CNF formula $\psi(\vec{x}, \vec{y})$ is

$$\varphi(\vec{x}) \leftrightarrow \exists \vec{y} \psi(\vec{x}, \vec{y}) \quad (2)$$

where $\exists \vec{y} \psi(\vec{x}, \vec{y})$ is a quantified Boolean formulas (will be introduced in Lecture02-06), which means that there exists a truth assignmet \vec{b} to \vec{y} such that $\psi(\vec{x}, \vec{b})$ holds, that it, $\bigvee_{\vec{b}} \psi(\vec{x}, \vec{b})$.

- If $\varphi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ satisfy (2), they also satisfy (1).

Proof(continued)

- We may assume that $\varphi(\vec{x})$ is in the negation normal form, that is, a formula with negation symbol \neg only in front of variables. This transformation by De Morgan's rule does not change the total number of variables, and the formula size at most double.
- Now, by induction on the construction of $\varphi(\vec{x})$, we construct a CNF formula $\psi(\vec{x}, \vec{y})$ satisfying (2).
- Suppose that for each $\varphi_i(\vec{x})$, there exists a CNF formula $\psi_i(\vec{x}, \vec{y}_i)$ satisfying (2).
- Then, for $\varphi(\vec{x}) \equiv \bigwedge_i \varphi_i(\vec{x})$, a CNF formula $\psi(\vec{x}, \vec{y}) \equiv \bigwedge_i \psi_i(\vec{x}, \vec{y}_i)$ also satisfies (2). Here, for $i \neq j$, \vec{y}_i and \vec{y}_j should not have common variables.

Proof(continued)

- For $\varphi(\vec{x}) \equiv \bigvee_i \varphi_i(\vec{x})$, we transform $\bigvee_i \psi_i(\vec{x}, \vec{y}_i)$ to a CNF formula:
 - ▷ Let $\psi'_1(\vec{x}, \vec{y}_1, z_1)$ be a CNF formula obtained from $\psi_1(\vec{x}, \vec{y}_1)$ by replacing each clause f with $f \vee z_1$.
 - ▷ For $i = 2, \dots, n-1$, let $\psi'_i(\vec{x}, \vec{y}_i, z_i, z_{i-1})$ be a CNF formula obtained from $\psi_i(\vec{x}, \vec{y}_i)$ by replacing each clause f with $f \vee z_i \vee \neg z_{i-1}$.
 - ▷ Let $\psi'_n(\vec{x}, \vec{y}_n, z_{n-1})$ be a CNF formula obtained from $\psi_n(\vec{x}, \vec{y}_n)$ by replacing each clause f with $f \vee \neg z_{n-1}$.
- Then $\bigwedge_i \psi'_i$ is a CNF and satisfies (2) with $\bigvee_i \psi_i(\vec{x}, \vec{y}_i)$.
- First suppose $\bigvee_i \psi_i(\vec{x}, \vec{y}_i) \in \text{SAT}$ and show $\bigwedge_i \psi'_i \in \text{SAT}$.
 - ▷ There exists some k such that $\psi_k(\vec{x}, \vec{y}_k) \in \text{SAT}$. Then ψ'_k is also satisfiable (regardless of the value of z_i).
 - ▷ Let z_1, \dots, z_{k-1} be T, and z_k, \dots, z_{n-1} be F. Then, for each $i \neq k$, $z_i \vee \neg z_{i-1}$ (z_1 for $i = 1$ and $\neg z_{n-1}$ for $i = n$) is T, and so ψ'_i is also T. Thus $\bigwedge_i \psi'_i \in \text{SAT}$.

Proof(continued)

- Conversely, suppose $\bigwedge_i \psi'_i \in \text{SAT}$.
 - ▷ Consider a truth value assignment that makes this formula \mathbb{T} and let z_k be the first variable that takes the value F . That is, from z_1 to z_{k-1} are \mathbb{T} , and z_k is F .
 - ▷ Since $z_k \vee \neg z_{k-1}$ added to each clause of ψ_k (although the form is slightly different when $k = 1, n$) is F , ψ_k must be \mathbb{T} to make ψ'_k satisfied. Thus, $\bigvee_i \psi_i \in \text{SAT}$.
- The size of $\bigwedge_i \psi'_i$ obtained in this way is a constant multiple of the total number of clauses contained in $\bigvee_i \psi_i$, and therefore is constant multiple of the input.
- In addition, the number of clauses does not change, and the number of repetition is less than the length of the input, so the size of the obtained CNF formula is bounded by n^2 for the size of the input n .
- Therefore, this transformation is a polynomial-time algorithm, and SAT is polynomial-time reducible to CNF-SAT. □

A CNF with exactly 3 literals in each clause is called a **3-CNF**, and the satisfaction problem for it is called **3-SAT**. It is also known to be NP-complete.

Theorem

3-SAT is NP-complete.

Proof.

- 3-SAT \in NP is obvious. Since CNF-SAT is NP-complete, it is enough to show CNF-SAT \leq_p 3-SAT.
- Let ϕ be a CNF formula. If there is a clause $f \equiv l_1 \vee \dots \vee l_k$ ($k \geq 4$) that contains 4 or more literals, replace it with the following formula g :

$$(l_1 \vee l_2 \vee x_1) \wedge (l_3 \vee \bar{x}_1 \vee x_2) \wedge (l_4 \vee \bar{x}_2 \vee x_3) \wedge \dots \wedge (l_{k-2} \vee \bar{x}_{k-4} \vee x_{k-3}) \wedge (l_{k-1} \vee l_k \vee \bar{x}_{k-3})$$

where \bar{x} represents $\neg x$.

- We can prove that the satisfiability of f is equivalent to the satisfiability of g , as in the second half of the proof of the last theorem.

Proof(continued)

- For a clause with only one literal l_1 , replace it with

$$(l_1 \vee x_1 \vee x_2) \wedge (l_1 \vee x_1 \vee \bar{x}_2) \wedge (l_1 \vee \bar{x}_1 \vee x_2) \wedge (l_1 \vee \bar{x}_1 \vee \bar{x}_2).$$

- For a clause with only two literals $l_1 \vee l_2$, replace it with

$$(l_1 \vee l_2 \vee x_1) \wedge (l_1 \vee l_2 \vee \bar{x}_1).$$

- It is easy to see that the satisfiability condition does not change by these transformations.
- Since the above transformations can be performed by polynomial-time algorithms, CNF-SAT is polynomial-time reducible to 3-SAT.



Homework

A CNF with exactly 2 literals in each clause is called a **2-CNF**, and the satisfaction problem for it is called **2-SAT**. Prove that 2-SAT belongs to P.

Summary

- We have defined the classes P and NP.
- Q_1 is polynomial (time) reducible to Q_2 , denoted as $Q_1 \leq_p Q_2$, if there exists a polynomial-time algorithm A which solves a problem q_1 in Q_1 as problem $A(q_1)$ in Q_2 .
- Q is NP-hard if for any NP problem Q' , $Q' \leq_p Q$.
- An NP-hard NP problem is said to be NP-complete.
- The Cook-Levin theorem: SAT is NP-complete.
- We have shown that CNF-SAT and 3-SAT is also NP-complete.

Further readings

M. Sipser, *Introduction to the Theory of Computation*, 3rd ed., Course Technology, 2012.

Recap

Historical
introduction

The classes P
and NP

Polynomial time
reducibility

NP-complete

Satisfiability
problem

Cook - Levin
theorem

CNF-SAT

3-SAT

Summary

Thank you for your attention!