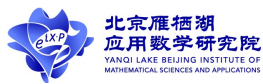
Logic and Computation: I

Chapter 2 Propositional logic and computational complexity

Kazuyuki Tanaka

BIMSA

December 1, 2022



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 graph
- Dec. 1, (5) Time-bound and space-bound complexity classes
- Dec. 6, (6) PSPACE-completeness and TQBF

Time-bound and space-bound complexity classes

Recap

- A Yes/No problem belongs to P if there exists a **deterministic** TM and a polynomial $p(x)$ s.t. for an input string of length n, it returns the correct answer within $p(n)$ steps.
- A problem belongs to NP if there is a **nondeterministic** TM and a polynomial $p(x)$ s.t. 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.
- Q_1 is polynomial (time) reducible to Q_2, denoted as $\mathsf{Q}_1 \leq_p \mathsf{Q}_2$, 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′, $\mathsf{Q}' \leq_p \mathsf{Q}$.
- An NP-hard NP problem is said to be NP-complete.

Theorem

The following are NP-complete: SAT, CNF-SAT, 3-SAT, VC, (d)HAMCYCLE, TSP.

Introduction

- In last lectures, we defined the P and NP classes by polynomial time constraints on Turing machines.
- Today, we will consider complexity classes defined by not only polynomials but also more general function families. We will also treat space (tape usage) constraints, and discuss their difference in computing power.
- The families of functions we treat as constraints are classified by asymptotic behavior.

Asymptotic notations

The followings are introduced to compare the growth rates of number-theoretic functions.

Definition

For number-theoretic functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$,

- $f(n) = O(g(n)) \overset{\text{def}}{\Leftrightarrow}$ there exists some $c > 0$ and for sufficiently large n,

$$f(n) \leq c \cdot g(n).$$

- $f(n) = \Theta(g(n)) \overset{\text{def}}{\Leftrightarrow} f(n) = O(g(n))$ and $g(n) = O(f(n))$.

- $f(n) = o(g(n)) \overset{\text{def}}{\Leftrightarrow}$ For any $c > 0$, for any sufficiently large n,

$$f(n) \leq c \cdot g(n).$$

Here, "for a sufficiently large n" means "there exists N s.t. for any $n \geq N$"; "$=$" is a special symbol, different from the usual equal symbol.

- The first "O" is particularly important, which is called the "Big O" notation.
- Note that in classical mathematics (Bachmann, Landau), "O" is often used in stead of "Θ".
- In addition to the above notation,

$$f(n) = \Omega(g(n)) \overset{\text{def}}{\Leftrightarrow} g(n) = O(f(n))$$

$$f(n) = \omega(g(n)) \overset{\text{def}}{\Leftrightarrow} g(n) = o(f(n))$$

are often used. But, we will not use them here, since it is easy to get confused with another usage: $f(n) = \Omega(g(n)) \Leftrightarrow$"for some $c > 0$ and infinitely many n, $f(n) \geq c \cdot g(n)$" (Hardy, Littlewood).

Unless otherwise stated, the base of the logarithmic function $\log(n)$ is 2, but for any $r > 1$, $\log_r(n) = \frac{1}{\log(r)} \log(n) = \Theta(\log(n))$.

Homework

- Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?
- Show $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.
- Show $\log(n!) = \Theta(n \log n)$.

- A number-theoretic function f used for bounding time and space is
  - ▷ monotonically increasing,
  - ▷ so simple (time-constructible and space-constructible) that it can be easily checked at any time during the computation whether it is in the time or space bound.

  By the latter condition, we may suppose that any computational process should halt within the time or space bound even if it is not accepted (as defined below).
- We assume that a Turing machine has one input tape and an arbitrary number of working tapes.
- We write $|x|$ for the length of the symbol string x.

Definition

- ▷ A (deterministic/non-deterministic) Turing machine runs in **time** $f(n)$ or is $f(n)$ **time-bounded**, if for every input x (except finitely many), its calculation process ends within $f(|x|)$ steps.
- ▷ A (deterministic/non-deterministic) Turing machine runs in **space** $f(n)$ or is $f(n)$ **space-bounded**, if for every input x (except finitely many), its calculation does not use more than $f(|x|)$ cells on each working tape.

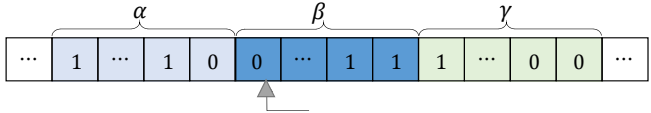
- We here consider a Turing machine M with time or space-bounding function. For a simple and contructible function, the machine can check by itself whether it is in the time or space bound. Otherwise, we need an outside system for measuring the time or space to use.
- As for the space bound, we only measure the used spaces of the working tapes. Therefore, the space bounding function $f(n)$ can take a value smaller than the input length n (e.g., $f(n) = \log n$).
- As for the time bound, it is usually necessary to finish reading all the input, so the time bounding function $f(n)$ will be more than $n + 1$.
- A complexity class is defined by a family of bounding functions classified by their asymptotic behavior.

- Before defining complexity classes, we briefly explain the following theorem.

The Linear Speedup Theorem

A language acceptable by a $f(n)$ time/space-bounded TM is also acceptable by $cf(n)$ time/space-bounded TM, for any constant $c > 0$.

- First, consider a Turing machine with space bound (deterministic or non-determ).
- Let c be an integer, and a Turing machine M that runs in space $cS(n)$. We construct a Turing machine M' that emulates it in space $S(n)$. To this end, divide each working tape of M into segments with length c, and treat each segment as one symbol.
- For speedup, M' also treats a segment of c symbols of M as one symbol. To guarantee the time reduction, let β be a segment where the head is placed, and α and γ be the left and right of β.



- A series of movements until the head of M moves to the leftmost of α or γ should be taken as one step of M'. If you need m steps for this, you may first divide the tape into segments of mc symbols.

In the following, we often omit "bound" or "bounded" for short. For instance, we just say a $f(n)$ time TM for a $f(n)$ time-bounded TM. Also by "in $O(f(n))$ time (space)", we mean "for some $g(n) = O(f(n))$, $g(n)$ time (space)".

Definition

For the function $f : \mathbb{N} \to \mathbb{N}$, we define the following four **complexity classes**.

$$\begin{aligned}
\mathrm{DTIME}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ time deterministic TM}\},\\
\mathrm{NTIME}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ time non-deterministic TM}\},\\
\mathrm{DSPACE}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ space deterministic TM}\},\\
\mathrm{NSPACE}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ space non-deterministic TM}\}.
\end{aligned}$$

By using $O(f(n))$ for bounding, we have a stable class that does not depend on the detailed definition of the Turing machine. For important number-theoretic functions f, we have the following classes.

Definition (Major Complexity Classes 1)

$$
\begin{aligned}
\text{L (or LOGSPACE)} &\stackrel{\text{def}}{=} \mathrm{DSPACE}(\log n),\\
\text{NL (or NLOGSPACE)} &\stackrel{\text{def}}{=} \mathrm{NSPACE}(\log n),\\
\text{P} &\stackrel{\text{def}}{=} \mathrm{DTIME}(n^{O(1)}) = \bigcup_k \mathrm{DTIME}(n^k),\\
\text{NP} &\stackrel{\text{def}}{=} \mathrm{NTIME}(n^{O(1)}) = \bigcup_k \mathrm{NTIME}(n^k),\\
\text{PSPACE} &\stackrel{\text{def}}{=} \mathrm{DSPACE}(n^{O(1)}) = \bigcup_k \mathrm{DSPACE}(n^k),\\
\text{NPSPACE} &\stackrel{\text{def}}{=} \mathrm{NSPACE}(n^{O(1)}) = \bigcup_k \mathrm{NSPACE}(n^k).
\end{aligned}
$$

Definition (Major Complexity Classes 2)

$$\begin{aligned}
\text{EXP (or EXPTIME)} &\stackrel{\text{def}}{=} \mathrm{DTIME}(2^{n^{O(1)}}) &&= \bigcup_k \mathrm{DTIME}(2^{n^k}),\\
\text{NEXP (or NEXPTIME)} &\stackrel{\text{def}}{=} \mathrm{NTIME}(2^{n^{O(1)}}) &&= \bigcup_k \mathrm{NTIME}(2^{n^k}),\\
\text{EXPSPACE} &\stackrel{\text{def}}{=} \mathrm{DSPACE}(2^{n^{O(1)}}) &&= \bigcup_k \mathrm{DSPACE}(2^{n^k}),\\
\text{NEXPSPACE} &\stackrel{\text{def}}{=} \mathrm{NSPACE}(2^{n^{O(1)}}) &&= \bigcup_k \mathrm{NSPACE}(2^{n^k}).
\end{aligned}$$

Although not introduced here, the class E $\stackrel{\text{def}}{=} \mathrm{DTIME}(2^{O(n)})$ and NE $\stackrel{\text{def}}{=} \mathrm{NTIME}(2^{O(n)})$ should not be confused with EXP and NEXP.

Example 1

- The problem $\mathbf{STCon}(\mathbf{nect})$ is to determine whether there is a path from s to t for two vertices s and t of a directed graph G.
- Non-deterministic space complexity: $\mathrm{STcon} \in \mathsf{NL}$.
  - Let n be the number of vertices of G. Extend a path from s non-deterministically, and accept it when it reaches t. Because it does not need to record the history, $2 \log n$ space is enough for the information on the end node of the current path and the next node. In order to terminate the algorithm, another $\log n$ space may be used to count the length of the path. So it is determined non-deterministically in $3 \log n$ space.
- For non-deterministic time complexity, $\mathrm{STCon}(\mathrm{nect})$ is roughly linear time (strictly, we need more rigorous definitions for machines and graphs).
- For deterministic case, STCon is in P and $\mathrm{DSPACE}(\log^2 n)$, which is shown by Theorem (2) and Savich's Theorem of this lecture.

We now examine the inclusion relationships between various complexity classes. Note that the functions f that define the major complexity classes are all constructible.

Time-constructible / Space-constructible functions

- $f(n)$ is **time-constructible** if there is a deterministic machine that count $f(n)$ in $O(f(n))$ steps with input 1^n.
- $f(n)$ is **space-constructible** if there is a deterministic machine that, for input 1^n, marks $S(n)$ cells and stops without using more than $S(n)$ cells of the working tape.

Example 2

- $\log n$, $(\log n)^2$ are space-constructible.
- n, $n \log n$, n^3, $2^{(\log n)^2}$, 2^n, $n!$, 2^{2n} are time and space constructible.

From now on, unless otherwise stated,

- $T(n)$ is a time-constructible number-theoretic function, and $T(n) > n$.
- $S(n)$ is a space-constructible number-theoretic function, and $S(n) \geq \log n$.

First, the following are clear from the definition of a (non-)deterministic Turing machine.

$$\begin{aligned} \mathrm{DTIME}(T(n)) &\subseteq \mathrm{NTIME}(T(n)), \\ \mathrm{DSPACE}(S(n)) &\subseteq \mathrm{NSPACE}(S(n)). \end{aligned}$$

The following are also clear from the fact that a machine can only move the head on each tape by one cell.

$$\begin{aligned} \mathrm{DTIME}(T(n)) &\subseteq \mathrm{DSPACE}(T(n)), \\ \mathrm{NTIME}(T(n)) &\subseteq \mathrm{NSPACE}(T(n)). \end{aligned}$$

Theorem (1)

For any space-constructible function $S(n) \geq \log n$,

$$\begin{aligned} \mathrm{DSPACE}(S(n)) &\subseteq \mathrm{DTIME}(2^{O(S(n))}) = \bigcup_k \mathrm{DTIME}(2^{kS(n)}), \\ \mathrm{NSPACE}(S(n)) &\subseteq \mathrm{NTIME}(2^{O(S(n))}) = \bigcup_k \mathrm{NTIME}(2^{kS(n)}). \end{aligned}$$

In particular, L $\subseteq$ P, and NL $\subseteq$ NP.

Proof.

- We only consider the deterministic case. The proof for the non-deterministic case is almost the same.
- Let M be a machine running in space $S(n)$. Assume M has only a single working tape with a separate track. We use the track for a clock so that it will stop in $c^{S(n)}$ steps.

Proof.(continued)

- Ω and Q are the set of symbols and of states for M. Let $|\Omega| = d$, and $|Q| = q$.
- For an input of length n, at most $d^{S(n)}$ sequences of symbols can be written on the working tape before stopping.
- A computational configuration of M is determined by such a sequence on the working tape together with a state, an input head position, a working head position. Thus, the number of configurations is $\leq qnS(n)d^{S(n)} \leq c^{S(n)}$ for a sufficiently large constant c ($\because S(n) \geq \log n$).
- So, a computational process longer than $c^{S(n)}$ includes a repetition of the same configuration. Hence, we may only consider computational processes shorter than this.
- With the counting track, the computation will stop in $c^{S(n)}$ whatever it accepts or not.
- Since the counter takes $O(S(n))$ steps for one step of mimicking M, the total time is $O(S(n)c^{S(n)})$, which is also $2^{O(S(n))}$. □

We have shown

$$\begin{aligned} \mathrm{NTIME}(T(n)) &\subseteq \mathrm{NSPACE}(T(n)) \\ \mathrm{NSPACE}(S(n)) &\subseteq \mathrm{NTIME}(2^{O(S(n))}) \end{aligned}$$

The following theorem is obtained from these relation by strengthening the right-hand sides to deterministic.

Theorem (2)

For any $T(n)$ and $S(n) \geq \log n$,

$$\begin{aligned} \mathrm{NTIME}(T(n)) &\subseteq \mathrm{DSPACE}(T(n)), \\ \mathrm{NSPACE}(S(n)) &\subseteq \mathrm{DTIME}(2^{O(S(n))}) \qquad (\clubsuit) \end{aligned}$$

In particular, NP $\subseteq$ PSPACE, and NL $\subseteq$ P.

Proof.

$\mathrm{NTIME}(T(n)) \subseteq \mathrm{DSPACE}(T(n))$

- Given a $T(n)$ time non-deterministic machine M, perform a depth-first search on its computation tree deterministically.
- The $T(n)$ space of M can be used repeatedly for calculation, and its track is also used to remember which calculation processes have been searched.

$\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$

- Imitate the non-deterministic machine M with width-first search deterministically.
- The proof is almost the same as that of Theorem (1).

Theorem (Savitch's theorem)

For any $S(n) \geq \log n$,

$$\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2).$$

In particular, PSPACE = NPSPACE, and EXPSPACE = NEXPSPACE.
By Example 1, $\mathrm{STcon} \in \mathrm{DSPACE}(\log^2(n))$.

Proof.

- By the proof of (♣) in Theorem (2), for a $S(n)$ space non-deterministic machine M , there exists some constant c such that M can be mimicked by a $c^{S(n)}$ time deterministic machine.
- Since $c^{S(n)}$ space is used for this imitation, we will improve as below.
- For $S(n)$ space machine M, the existence of a transition from a configuration α to a configuration β within k steps is represented by $\mathrm{Reach}(\alpha, \beta, k)$.
- Next we determine $\mathrm{Reach}(\alpha, \beta, k)$ recursively.
  - ▷ If $k = 0$, check whether $\alpha = \beta$.
  - ▷ If $k = 1$, check whether it can move from α to β in one step.
  - ▷ If $k \geq 2$, check whether there is a computational configuration γ that satisfies both $\mathrm{Reach}\left(\alpha, \gamma, \frac{k}{2}\right)$ and $\mathrm{Reach}\left(\gamma, \beta, \frac{k}{2}\right)$. If so, $\mathrm{Reach}(\alpha, \beta, k)$ also holds.

    If $\frac{k}{2}$ is not an integer, one side is rounded up and the other rounded down.

Proof(continued).

- ▷ For $\frac{k}{2} \geq 2$, first check the existence of γ' such that $\mathrm{Reach}\left(\alpha, \gamma', \frac{k}{2^2}\right)$ and $\mathrm{Reach}\left(\gamma', \gamma, \frac{k}{2^2}\right)$.
- ▷ Then check the existence of γ' such that $\mathrm{Reach}\left(\gamma, \gamma', \frac{k}{2^2}\right)$ and $\mathrm{Reach}\left(\gamma', \beta, \frac{k}{2^2}\right)$.

- By repeating recursive branchings in this way, we obtain a binary tree with a height of about $\log_2 k = O(S(n))$. In each stage, $O(S(n))$ space is necessary to memorize the configuration. In total, it can be executed in $O(S(n)^2)$ space.

□

The following theorem due to N. Immerman[1] and R. Szelepcsényi[2] gives an evidence that the space complexity class is easier to handle than the time complexity class.

Theorem (Immermann- Szelepcsényi theorem)

For any $S(n)(\geq \log n)$, $\mathrm{NSPACE}(S(n))$ is closed under complement.

Proof.

- Suppose a nondeterministic machine M accepts a language A with $S(n)$ space, we will construct a nondeterministic machine $\overline{M}$ that accepts the complement A^c with $S(n)$ space.
- A configuration of M can be represented by a string of length $S(n)$, and the total number is $c^{S(n)}$ for some constant c.

[1] N. Immerman, Nondeterministic space is closed under complementation, SIAM Journal on Computing 17 (1988), 935-938.

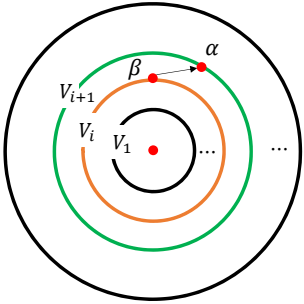
[2] R. Szelepcsény, The method of forced enumeration for nondeterministic automata, Acta Informatica 26 (1988), 279-284.

Proof(continued).

- Consider the directed graph G with the configurations as vertices and the transition relations as directed edges.
- It is sufficient to determine whether there is a path from the initial state to the final state in the graph G. Note that we may assume that M has a unique accepting configuration, by making M erase its worktape and return its heads to the starting positions after the computation and then entering a unique accept state.
- To check whether the answer is yes, it is possible in $\log(c^{S(n)}) = O(S(n))$ space as shown in Example 1.
- To get No, it is necessary to exhaustively examine all possible paths. It seems to require a huge space at first glance, but there is a surprisingly simple process. By counting the number $m(i)$ of vertices that can be reached within i steps from the initial state, it is possible to check whether all paths are examined.

Proof(continued).

- Let V_i be the set of vertices reachable within i steps from the initial configuration. Let $m(i) = |V_i|$.
- $\alpha \in V_{i+1}$ iff there exists $\beta \in V_i$ such that there is edge from β to α.
- To compute $m(i+1)$, we successively check a vertex $\alpha \in V_{i+1}$ in some order. If so, increment a counter by one. The final value of the counter is $m(i+1)$.



- To check a vertex $\alpha \in V_{i+1}$, non-deterministically guess $m(i)$ elements β of V_i in some order and check if there is an edge from β to α.
- Finally, for some $i \leq c^{S(n)}$, $m(i+1) = m(i)$.
- If the final configuration does not appear before i, i.e., if the final configuration does not enter V_i, then M will not accept the input, so $\overline{M}$ will accept it.

□

Since $\mathrm{NSPACE}(n)$ matches the class of context-sensitive languages, this also solved the long-standing open question: Is the complement of a context-sensitive language is still context-sensitive?

Highlights of the above results are:

$$\mathsf{PSPACE} = \mathsf{NPSPACE}, \qquad \mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$$

and

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$$

Among them, topic on which is/are a proper inclusion relation will be discussed in the next lecture.

Summary

- For the function $f : \mathbb{N} \to \mathbb{N}$, we define the following four **complexity classes**.

$$\begin{aligned}
\mathrm{DTIME}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ time deterministic TM}\},\\
\mathrm{NTIME}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ time non-deterministic TM}\},\\
\mathrm{DSPACE}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ space deterministic TM}\},\\
\mathrm{NSPACE}(f(n)) &\stackrel{\mathrm{def}}{=} \{L(M) \mid M \text{ is } O(f(n)) \text{ space non-deterministic TM}\}.
\end{aligned}$$

- For major functions f, we have

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$$

- Savitch' theorem: for any $S(n) \geq \log n$, $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$.
- Immermann-Szelepcsényi's theorem: for any $S(n)(\geq \log n)$, $\mathrm{NSPACE}(S(n))$ is closed under complement.

Further readings

D.C. Kozen, *Theory of Computation*, Springer, 2006.

Thank you for your attention!