Logic and Computation I

Chapter 2. Propositional logic and computational complexity

Kazuyuki Tanaka

BIMSA

October 24, 2024

北京雁栖湖应用数学研究院
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 4. Modal logic**

Part 2. Schedule

- Oct.10, (1) Tautologies and proofs
- Oct.15, (2) The completeness theorem of propositional logic
- Oct.17, (3) SAT and NP-complete problems
- Oct.22, (4) NP-complete problems about graphs
- Oct.24, (5) Time-bound and space-bound complexity classes
- Oct.29, (6) PSPACE-completeness and TQBF

Recap

- An NP-hard NP problem is said to be NP-complete.
- The following are NP-complete: SAT, $\mathrm{CNF\text{-}SAT}$, $\mathrm{3\text{-}SAT}$, VC and $\mathrm{(d)HAMCYCLE}$.

TSP

The **Traveling Salesman Problem**: does there exist a Hamiltonian cycle such that the sum of edge weights is less than or equal to k?

It can be shown that TSP is also NP-complete.

- For TSP to be NP, choose an arbitrary path and check whether it satisfies the condition or not.
- For the reversal, the existence of a Hamiltonian cycle is the existence of a TSP solution with edge weight 1 and sufficiently large k, and so

$$\mathrm{HAMCYCLE} \leq_p \mathrm{TSP}.$$

§2.5. Time/space-bound complexity classes: 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 used to compare the growth rates of number-theoretic functions.

Definition 2.31

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 also 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 $r > 1$, $\log_r(n) = \frac{1}{\log(r)}\log(n) = \Theta(\log(n))$.

Exercise 2.5.1

- 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 checked at any time during 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.

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 2.32

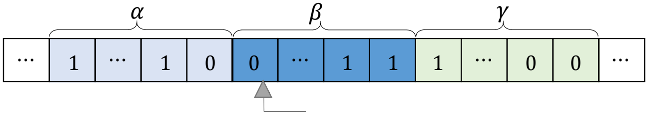
- ▷ A (deterministic/non-deterministic) Turing machine runs in **time** $f(n)$ or is $f(n)$ **time-bounded**, if for every input x (except finitely many), its calculation process ends within $f(|x|)$ steps.
- ▷ A (deterministic/non-deterministic) Turing machine runs in **space** $f(n)$ or is $f(n)$ **space-bounded**,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

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., $f(n) = \log n$).

The Linear Speedup Theorem

A language acceptable by a $f(n)$ time/space-bounded TM (det or non-det) is also acceptable by $\epsilon f(n)$ time/space-bounded TM, for any constant $\epsilon > 0$.

- Let $c > 0$ 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. Let β be a segment where the head is placed, and α and γ be the left and right of β. In 4 steps (L,R,R,L), M' can gather all information of α, β, γ.



- During c consecutive moves of M, its head will stay in (α, β) or (β, γ). So in 2 steps, M' can change its tape according to M's configuration after c moves. In sum, M' can mimic M's c moves in 6 steps, that is, it is $\frac{6}{c} f(n)$ time bound.

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 2.33

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

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

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 2.34 (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 2.34 (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 4

- The problem $\mathbf{STCon(nect)}$ 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: $\mathrm{STCon} \in$ 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, $O(\log n)$ space is enough for keeping the information on the current vertex, the next one you will visit, and the number of steps you have taken.
- The above NL algorithm can been roughly seen as a non-deterministic linear time one.
- For deterministic case, STCon is in P and $\mathrm{DSPACE}(\log^2 n)$, which is shown by Theorem 2.36 and 2.37 (Savitch's Theorem).

We now examine the inclusion relationships between various complexity classes.

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 for input 1^n.
- $S(n)$ 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.

Examples

- $\log n$, $(\log n)^2$ 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 2.35

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. Then a machine M' mimicking M in time $2^{O(S(n))}$ has also one working tape but with a separate track for a clock in order to stop in $c^{S(n)}$ steps. c will be given below.

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.
- M' mimicks M by updating M's configurations, so it takes $O(S(n))$ steps for mimicking one step of M. M' also needs some steps for updating the counting track, but they can be also included in $O(S(n))$.
- With the counting track, M''s simulation will stop in $c^{S(n)}$, so the total time for M' is $O(S(n)c^{S(n)})$, which is $2^{O(S(n))}$. $\square$

Theorem 2.36

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

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

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

Proof.

($\diamondsuit$) Given a $T(n)$-time NTM M, a DTM M' performs a depth-first search on its computation tree. M' does not need to remember the configuration history of M, but only needs which calculation processes have been searched. So, $T(n)$ space can be used repeatedly for calculation.

($\clubsuit$) A DTM M' imitates a $S(n)$-space NTM M with width-first search in a way similar to that of Theorem 2.35.

Theorem 2.37 (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 4, $\mathrm{STCon} \in \mathrm{DSPACE}(\log^2(n))$.

Proof.

- By the proof of (♣) in Theorem 2.36, for a $S(n)$-space NTM M , there exists some constant c such that M can be mimicked by a $c^{S(n)}$-time DTM. This simulation needs $c^{S(n)}$ space, but can be improved as below.
- For a $S(n)$-space NTM M, the existence of a transition from a configuration α to a configuration β within $k(\leq c^{S(n)})$ steps is represented by $\mathrm{Reach}(\alpha, \beta, k)$.

Proof(continued). $\mathrm{Reach}(\alpha, \beta, k)$ is determined recursively as follows.

- 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.
- For $\frac{k}{2} \geq 2$, first seek a γ' s.t. $\mathrm{Reach}\left(\alpha, \gamma', \frac{k}{2^2}\right)$ and $\mathrm{Reach}\left(\gamma', \gamma, \frac{k}{2^2}\right)$.

  Later, seek a γ' s.t. $\mathrm{Reach}\left(\gamma, \gamma', \frac{k}{2^2}\right)$ 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. $\square$

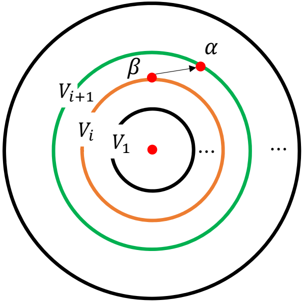
Theorem 2.38 (Immermann- Szelepcsényi theorem)

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

Proof.

- Suppose a NTM M accepts a language A with $S(n)$ space, we will construct a NTM $\overline{M}$ 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 so the total number is $c^{S(n)}$ for some constant c.
- 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 a final state in the graph G. 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.
- To get the answer Yes for the existence of such a path, it is computable in $\log(c^{S(n)}) = O(S(n))$ space as shown in Example 4 (STCon).

- To get No, it seems to require a huge amount of spaces to examine all possible paths, but there is a surprisingly simple process.
- Let V_i be the set of vertices reachable within i steps from the initial configuration. Let $m(i) = |V_i|$. By counting $m(i)$, we can check whether all paths are examined.
- To compute $m(i+1)$, we check whether $\alpha \in V_{i+1}$ for each vertex α in some order. If yes, 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 choose an element β of V_i (with a possible i-step path from the initial conf.) and check if there is an edge from β to α or $\alpha = \beta$. Iterate this process $m(i)$ times.
- 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 a function $f : \mathbb{N} \to \mathbb{N}$, we define the following four **complexity classes**.

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

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

- 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!