Theory of Computation

Contents

§ 1. Preliminaries 1
§ 2. Computability Theory 2
2.1. Turing Machines 2
2.2. Language Classes 4
2.3. Church–Turing Thesis 6
2.4. Universal Turing Machine 6
2.5. The Halting Problem and Reductions 6
2.6. The Arithmetical Hierarchy 8
§ 3. Computational Complexity Theory 9
3.1. Deterministic Classes 9
3.2. Nondeterministic Turing Machines 10
3.3. Sparse Languages and the Structure of NP 11

§ 1. Preliminaries

> **Definition.** A decision problem is a pair $\Pi = (D_\Pi, Y_\Pi)$, where D_Π is the set of problem instances and $Y_\Pi \subseteq D_\Pi$ is the set of "YES" instances.

> **Definition.** An alphabet Σ is any finite, non-empty set of symbols. In particular, if $|\Sigma| = 1$, we call the alphabet **unary**.

> **Definition.** A word (or string) over an alphabet Σ is any finite sequence of symbols from Σ. In particular, we denote the empty word by ε.

> **Definition.** A language L over an alphabet Σ is any set of words over Σ.

If A and B are languages, then:

- $A \cup B = \{x \mid x \in A \vee x \in B\}$ is the union of languages,
- $A \cap B = \{x \mid x \in A \wedge x \in B\}$ is the intersection of languages,
- $AB = \{xy \mid x \in A \wedge y \in B\}$ is the concatenation of languages,
- $A^* = \{\varepsilon\} \cup A \cup A^2 \cup \cdots$ is the Kleene closure (star) of A.

In particular, Σ^* denotes the set of all words over Σ. We also define $\Sigma^+ := \Sigma\Sigma^*$ and write $L' := \Sigma^* \setminus L$ for the complement of a language L.

Every language L defines a decision problem of the form: *does $x \in \Sigma^*$ belong to L?* Conversely, every "reasonable" decision problem defines a language, assuming there is an "effective" encoding function $\langle\cdot\rangle$ mapping problem instances to strings over Σ.

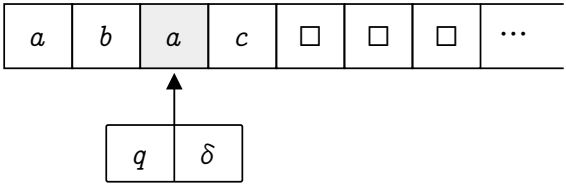
§ 2. Computability Theory

2.1. Turing Machines

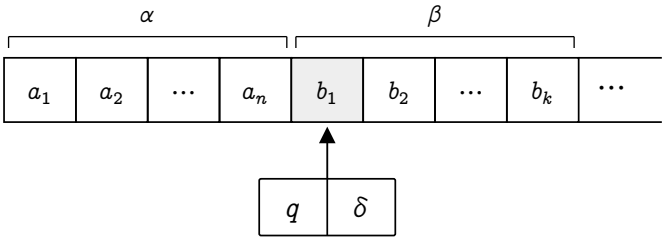
Definition. A Turing Machine M is a tuple $(\Sigma, \Gamma, Q, \delta, q_S, q_Y, q_N, \square)$, where:
- Σ — input alphabet,
- Γ — tape alphabet, with $\Sigma \subseteq \Gamma$,
- $\square$ — special blank symbol, assuming $\square \in \Gamma$ and $\square \notin \Sigma$,
- Q — finite, non-empty set of states,
- $q_S \in Q$ — start state,
- $q_Y \in Q$ — accepting state,
- $q_N \in Q$ — rejecting state ($q_Y \neq q_N$),
- $\delta : \Gamma \times (Q \setminus \{q_Y, q_N\}) \to \Gamma \times Q \times \{\leftarrow, \rightarrow\}$ — transition function.

We can picture a Turing Machine as consisting of a tape infinite to the right, divided into cells, together with a read/write head holding a state q that reads and writes symbols from Γ at the cell it currently points to. Initially, the input word $w \in \Sigma^*$ is written at the beginning of the tape, and all remaining cells contain the

blank symbol $\square$. When the head is in state q and reads $a \in \Gamma$, it writes b, transitions to state q', and moves left or right according to $\delta(a, q) = (b, q', \leftarrow \text{ or } \rightarrow)$.



Definition. A configuration of a machine M is any string of the form $\alpha q \beta$, where $\alpha \in \Gamma^*$, $q \in Q$, and $\beta \in \Gamma^+$, omitting the infinitely many trailing blank symbols in β. This notation means that the head points to the first symbol of $\beta = b_1 \cdots b_k$, to the left of which the tape holds $\alpha = a_1 \cdots a_n$, and the machine is in state q.



Definition. We say that a machine M accepts a word $x = x_1 \cdots x_n \in \Sigma^*$ if and only if there exists a finite sequence of configurations starting with $\varepsilon q_S x$ (or $\varepsilon q_S \square$ when $x = \varepsilon$), ending with $\alpha q_Y \beta$ for some $\alpha \in \Gamma^*$ and $\beta \in \Gamma^+$, such that for every step t the machine can transition from configuration k_t to k_{t+1} in a single step. If M neither accepts nor enters q_N on x, we say that M rejects x by looping forever.

Definition. We say that a machine M accepts (recognizes) a language L if and only if for every $x \in \Sigma^*$, M accepts x if and only if $x \in L$.

Definition. We say that a machine M decides a language L if and only if M accepts L and halts on every input word.

2.2. Language Classes

For a given machine M, let $L(M)$ denote the language accepted by M. We define the following classes of languages:

- $\mathrm{R} \coloneqq \{L \subseteq \Sigma^* \mid \exists M : M \text{ decides } L\}$,
- $\mathrm{RE} \coloneqq \{L \subseteq \Sigma^* \mid \exists M : M \text{ accepts } L\}$,
- $\mathrm{coRE} \coloneqq \{L' \mid L \in \mathrm{RE}\}$.

Languages in RE are called *recursively enumerable* (or *recognizable*), and languages in R are called *decidable* (or *recursive*). Notice that every finite language $L = \{x_1, \cdots, x_n\}$ is decidable, since we can build a Turing Machine that decides it:

Machine: $M(x)$ for a finite language $L = \{x_1, \cdots, x_n\}$

---

if $x = x_1$: **accept**
elif $x = x_2$: **accept**
$\vdots$
elif $x = x_n$: **accept**
else: **reject**

Note that this construction fails when L is infinite, because the finite description of the Turing Machine could not list infinitely many words.

Theorem. $\mathrm{RE} \cap \mathrm{coRE} = \mathrm{R}$.

Proof. Let $L \subseteq \Sigma^*$ be any language over Σ. First suppose that $L \in \mathrm{RE} \cap \mathrm{coRE}$. By definition, there exist machines M_L and $M_{L'}$

accepting L and L', respectively. We construct a machine M that decides L:

Machine: $M(x)$ deciding $L \in \mathrm{RE} \cap \mathrm{coRE}$

---

for $k = 1, 2, \cdots$:

> **if** $M_L(x)$ accepts in at most k steps: **accept**
> **if** $M_{L'}(x)$ accepts in at most k steps: **reject**

Every word $x \in \Sigma^*$ is accepted by exactly one of M_L and $M_{L'}$. Hence, for sufficiently large k, exactly one condition holds and M halts with the correct answer, accepting x if and only if $x \in L$. Thus $\forall L \subseteq \Sigma^* : L \in \mathrm{RE} \cap \mathrm{coRE} \implies L \in \mathrm{R}$.

Conversely, suppose that $L \in \mathrm{R}$, so there exists a machine M that decides L. We construct machines M_L and $M_{L'}$ accepting L and L':

Machine: $M_L(x)$ and $M_{L'}(x)$ for $L \in \mathrm{R}$

$M_L(x)$:

> **if** $M(x)$ accepts: **accept**
> **else**: **reject**

$M_{L'}(x)$:

> **if** $M(x)$ accepts: **reject**
> **else**: **accept**

Since M halts on every input x and accepts exactly the words in L, M_L accepts L and $M_{L'}$ accepts L'. Therefore $\forall L \subseteq \Sigma^* : L \in \mathrm{R} \implies L \in \mathrm{RE} \wedge L \in \mathrm{coRE} \iff L \in \mathrm{RE} \cap \mathrm{coRE}$, completing the proof. $\square$

2.3. Church–Turing Thesis

> **The formal notion of a Turing Machine is equivalent to the intuitive notion of an algorithm.**

2.4. Universal Turing Machine

A Universal Turing Machine U takes as input an encoded machine M together with an input word x, written $\langle M, x \rangle$, and simulates the execution of M on x. It is the theoretical model of a stored-program computer. Fixing the alphabet $\Sigma = \{0, 1, \#\}$, we can encode M as

$$\langle M \rangle = \langle \Gamma \rangle \ \# \ \langle Q \rangle \ \# \ \langle \delta \rangle \ \#\# \quad , \tag{2.1}$$

where $\langle \Gamma \rangle = \langle |\Gamma| \rangle$ and $\langle Q \rangle = \langle |Q| \rangle$ are binary encodings of the tape-alphabet size and the number of states, and

$$\begin{aligned} \langle \delta \rangle = \langle &\text{sequence of encoded transitions } \langle \delta(a, q) \rangle \\ &\text{for } q \in Q \setminus \{q_Y, q_N\}, a \in \Gamma \rangle \quad . \end{aligned} \tag{2.2}$$

2.5. The Halting Problem and Reductions

We define the halting problem as the language

$$H \coloneqq \{\langle M, x \rangle \mid M \text{ halts on } x\} \tag{2.3}$$

(similarly, the acceptance problem $A_{\mathrm{TM}} \coloneqq \{\langle M, x \rangle \mid M \text{ accepts } x\}$ shares the same decidability and completeness properties below).

> **Theorem.** The halting problem is undecidable, $H \notin \mathrm{R}$.

Theorem (Rice's Theorem). Let $\mathcal{L} \subseteq \text{RE}$ be a subset of RE. We say that a language $L \in \text{RE}$ has property $\mathcal{L}$ if and only if $L \in \mathcal{L}$. Whenever $\mathcal{L} \neq \emptyset$ and $\mathcal{L} \neq \text{RE}$, the language

$$B_{\mathcal{L}} \coloneqq \{\langle M \rangle \mid L(M) \in \mathcal{L}\} \tag{2.4}$$

is undecidable.

This theorem states that every non-trivial semantic property of the language recognized by a Turing Machine is undecidable.

Definition. Let $f : \Sigma^* \to \Sigma^*$. We say that f is computable if and only if there exists a Turing Machine M that, on every input $x \in \Sigma^*$, halts with $f(x)$ written on the tape starting at the head (configuration $\varepsilon q_Y f(x)$, or $\varepsilon q_Y \square$ when $f(x) = \varepsilon$).

Definition. We say that a language A reduces to a language B, written $A \leq_m B$, if and only if there exists a computable function f such that

$$\forall x \in \Sigma^* : x \in A \iff f(x) \in B \quad . \tag{2.5}$$

The subscript m denotes a *many-to-one* reduction, meaning multiple (even all) words from A may map to the same word in B. If $A \leq_m B$, then A is no harder than B: any solver for B immediately yields a solver for A by first applying f. Key properties of reductions:

- if $A \leq_m B$ and $B \in \text{R}$, then $A \in \text{R}$,
- if $A \leq_m B$ and $B \in \text{RE}$, then $A \in \text{RE}$,
- if $A \leq_m B$ and $A \notin \text{R}$, then $B \notin \text{R}$.

Definition. A language B is RE-hard if and only if $A \leq_m B$ for every language $A \in \text{RE}$.

Definition. A language B is RE-complete if and only if it is RE -hard and $B \in \text{RE}$.

Theorem. The halting language H is RE-complete, $H \in$ RE-com.

The standard template for proving that a language C is RE-complete is therefore:

- take a known RE-complete language $B \in$ RE-com,
- show that $C \in$ RE,
- show that $B \leq_m C$; by transitivity of $\leq_m$, for every $A \in$ RE we have $A \leq_m B \leq_m C$, which together with $C \in$ RE establishes $C \in$ RE-com.

Theorem. A language A belongs to RE if and only if there exists a decidable language $B \in$ R such that

$$A = \{x \mid \exists y \in \Sigma^* : \langle x, y \rangle \in B\} \quad . \tag{2.6}$$

2.6. The Arithmetical Hierarchy

Definition. A language $A \subseteq \Sigma^*$ belongs to the class Σ_i for $i \geq 1$ if and only if there exists $B \in$ R such that

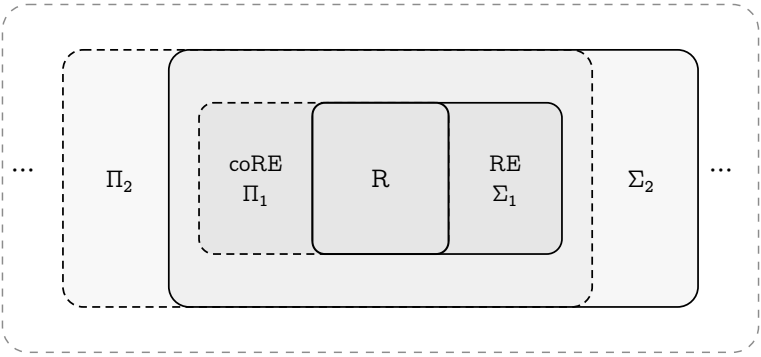
$$A = \{x \mid \exists y_1 : \forall y_2 : \cdots : Q_i y_i : \langle x, y_1, \cdots, y_i \rangle \in B\} \quad , \tag{2.7}$$

where $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even. Furthermore, A belongs to Π_i if and only if $A' \in \Sigma_i$.

From this definition and the preceding theorem it follows that $\Sigma_1 =$ RE and $\Pi_1 =$ coRE. These classes form an infinite hierarchy satisfying

$$\Sigma_{i-1} \subseteq \Sigma_i \cap \Pi_i \quad , \quad \Pi_{i-1} \subseteq \Sigma_i \cap \Pi_i \quad , \tag{2.8}$$

where $i \geq 1$ and $\Sigma_0 = \Pi_0 =$ R.



§ 3. Computational Complexity Theory

3.1. Deterministic Classes

Definition. Let M be a Turing Machine. The time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$ giving the maximum number of steps taken by M over all inputs x of length $|x| = n$.

Definition. Let M be a Turing Machine. The space complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$ giving the maximum number of tape cells visited by M over all inputs x of length $|x| = n$.

Definition. $\mathrm{TIME}(t(n))$ is the set of languages L for which there exists a Turing Machine M deciding L in time $O(t(n))$.

Definition. $\mathrm{SPACE}(s(n))$ is the set of languages L for which there exists a Turing Machine M deciding L in space $O(s(n))$.

From these definitions we construct the standard deterministic complexity classes:

$$
\begin{aligned}
\mathrm{P} &= \bigcup_{k=0}^{\infty} \mathrm{TIME}\left(n^k\right) \quad , \\
\mathrm{EXP} &= \bigcup_{k=0}^{\infty} \mathrm{TIME}\left(2^{n^k}\right) \quad , \\
\mathrm{PSPACE} &= \bigcup_{k=0}^{\infty} \mathrm{SPACE}\left(n^k\right) \quad , \\
\mathrm{EXPSPACE} &= \bigcup_{k=0}^{\infty} \mathrm{SPACE}\left(2^{n^k}\right) \quad .
\end{aligned}
\tag{3.1}
$$

3.2. Nondeterministic Turing Machines

Definition. A Nondeterministic Turing Machine (NTM) is a tuple $(\Sigma, \Gamma, Q, \delta, q_S, q_Y, q_N, \square)$, defined identically to a deterministic Turing Machine except that δ is now a transition relation

$$
\delta \subseteq (\Gamma \times (Q \setminus \{q_Y, q_N\})) \times (\Gamma \times Q \times \{\leftarrow, \rightarrow\}) \quad . \tag{3.2}
$$

Reading a symbol in a given state, an NTM may:

- have exactly one valid transition,
- have more than one valid transition,
- have no valid transition (treated as a rejecting configuration).

Definition. An NTM M accepts a word x in at most k steps if and only if:

- every computation branch on x takes at most k steps,
- at least one branch accepts in at most k steps.

Definition. An NTM M rejects a word x in at most k steps if and only if:

- every computation branch on x takes at most k steps,
- every branch rejects in at most k steps.

Definition. Let $f : \mathbb{N} \to \mathbb{N}$. An NTM M decides L in time $f(n)$ if and only if M accepts $x \in \Sigma^*$ if and only if $x \in L$, and for

every $x \in \Sigma^*$ all computation branches of M on x have length at most $f(|x|)$.

Definition. $\mathrm{NTIME}(t(n))$ is the set of languages L for which there exists an NTM M deciding L in time $O(t(n))$.

Analogously to the deterministic classes, we define:

$$\begin{aligned} \mathrm{NP} &= \bigcup_{k=0}^{\infty} \mathrm{NTIME}\left(n^k\right) \quad , \\ \mathrm{NEXP} &= \bigcup_{k=0}^{\infty} \mathrm{NTIME}\left(2^{n^k}\right) \quad . \end{aligned} \tag{3.3}$$

3.3. Sparse Languages and the Structure of NP

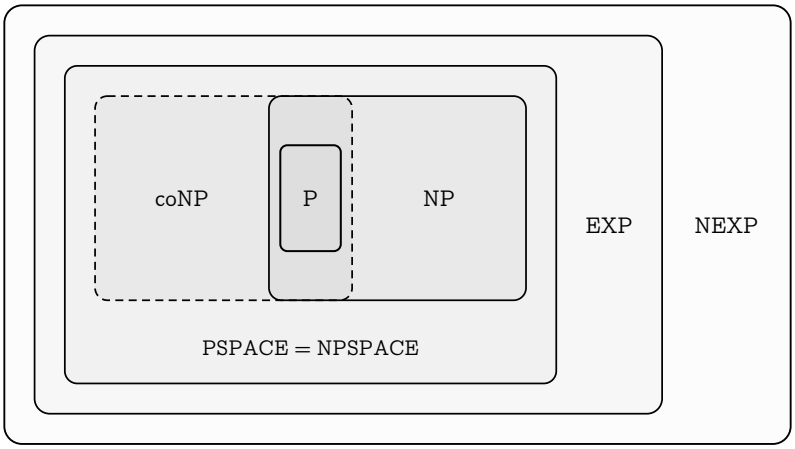
Definition. A language L is *sparse* if and only if there exists a polynomial p such that $\forall n \in \mathbb{N} : |L^{\leq n}| \leq p(n)$, where $L^{\leq n} = \{x \in L \mid |x| \leq n\}$.

Definition. A language L is *unary* if and only if $L \subseteq \{0\}^*$.

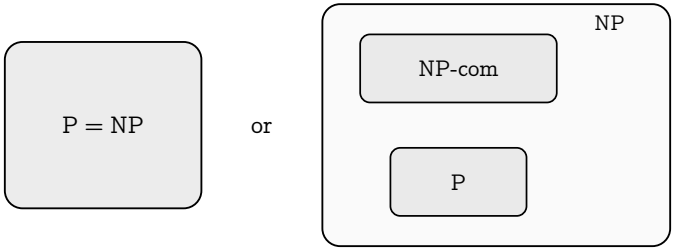
Theorem. Every unary language is sparse.

Theorem (Mahaney, 1982). If there exists a sparse language that is NP-hard under polynomial-time many-to-one reductions ($\leq_m^p$), then $\mathrm{P} = \mathrm{NP}$.

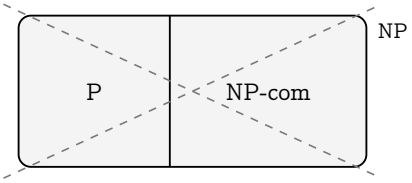
The inclusions among the main complexity classes are summarized below:



In particular, the internal structure of NP can only look like one of the following two pictures:



and cannot look like this:



Theorem. If $\mathrm{P} \neq \mathrm{NP}$, then there exists a language A such that $A \in \mathrm{NP}$, $A \notin \mathrm{P}$, and $A \notin \mathrm{NP\text{-}com}$.

Theorem. If $\mathrm{P} = \mathrm{NP}$, then $\mathrm{EXP} = \mathrm{NEXP}$.

Theorem (Ladner, 1975). For every decidable language $B \in \mathrm{R} \setminus \mathrm{P}$, there exists a language $A \in \mathrm{P}$ such that:

- $A \cap B \notin \mathrm{P}$,
- $A \cap B \leq_m^p B$,
- $B \not\leq_m^p A \cap B$.

Here $\leq_m^p$ denotes a polynomial-time many-to-one reduction, i.e., a many-to-one reduction whose reducing function is computable in polynomial time.