Lecture Notes

Complexity Theory

taught in Winter term 2016

by

Sven Kosub

February 15, 2017

Version v2.11

Contents

Complexity measures and classes 1

1.1 Deterministic measures

1.1.1 A general framework

We introduce a general, notational framework for complexity measures and classes.

Let τ be an algorithm *type*, e.g., Turing machine, RAM, Pascal/C/Java program, etc. For any algorithm of type τ, it must be defined when the algorithm *terminates* (stops, halts) on a given input and, if the algorithm terminates, what the *result* (outcome/output) is.

An algorithm A of type τ *computes* a mapping $\varphi_A : (\Sigma^*)^m \to \Sigma^*$:

$$\varphi_A(x) =_{\text{def}} \begin{cases} \text{result of } A \text{ on input } x & \text{if } A \text{ terminates on input } x \\ \text{not defined} & \text{otherwise} \end{cases}$$

A *complexity measure* for algorithms of type τ is a mapping Φ:

$$\Phi \ : \ \text{finite computation of } A \text{ of type } \tau \text{ on input } x \ \mapsto \ r \in \mathbb{N}$$

A (finite) computation is a (finite) sequence of configurations (specific to the type τ). Note that, in general, such a sequence need not be complete in the sense that it is reachable from a defined initial configuration.

Examples: Standard complexity measures are the following:

$$\Phi = \tau\text{-DTIME} \qquad \Phi = \tau\text{-DSPACE}$$

Here, "D" indicates that algorithms are deterministic.

A *complexity function* for an algorithm A of type τ is a mapping $\Phi_A : (\Sigma^*)^m \to \mathbb{N}$:

$$\Phi_A(x) =_{\text{def}} \begin{cases} \Phi(\text{computation of } A \text{ on input } x) & \text{if } A \text{ terminates on input } x \\ \text{not defined} & \text{otherwise} \end{cases}$$

A *worst-case complexity function* of A of type τ is a mapping $\Phi_A : \mathbb{N} \to \mathbb{N}$:

$$\Phi_A(n) =_{\text{def}} \max_{|x|=n} \Phi_A(x)$$

Resource bounds are compared asymptotically. Define for $f, g : \mathbb{N} \to \mathbb{N}$:

$$f \leq_{\text{ae}} g \iff_{\text{def}} (\exists n_0)(\forall n)[n \geq n_0 \to f(n) \leq g(n)]$$

The subscript "ae" refers to "almost everywhere."

Let $t : \mathbb{N} \to \mathbb{N}$ be a *resource bound* (i.e., t is monotone). We say that an algorithm A of type τ *computes* the total function f *in* Φ*-complexity* t if and only if $\varphi_A = f$ and $\Phi_A \leq_{\text{ae}} t$. We define the following complexity classes:

$$\begin{aligned}
\mathrm{F}\Phi(t) &=_{\text{def}} \{\ f \mid f \text{ is a total function and there is an algorithm } A \text{ of type } \tau \\
&\qquad\qquad\qquad\qquad \text{computing } f \text{ in } \Phi\text{-complexity } t\ \} \\
\mathrm{F}\Phi(O(t)) &=_{\text{def}} \bigcup_{k\geq 1} \mathrm{F}\Phi(k \cdot t) \\
\mathrm{F}\Phi(\mathrm{Pol}\ t) &=_{\text{def}} \bigcup_{k\geq 1} \mathrm{F}\Phi(t^k)
\end{aligned}$$

When considering the complexity of languages (sets) instead of functions, we use the characteristic function to define the fundamental complexity classes. The *characteristic function* $c_L : \Sigma^* \to \{0,1\}$ of a language $L \subseteq \Sigma^*$ is defined to be for all $x \in \Sigma^*$:

$$c_L(x) = 1 \iff_{\text{def}} x \in L$$

Recall that an algorithm A accepts (decides) a language $L \subseteq \Sigma^*$ if and only if A computes c_L. Accordingly, we say that A *accepts* a language L *in* Φ*-complexity* t if and only if $\varphi_A = c_L$ and $\Phi_A \leq_{\text{ae}} t$. We obtain the following deterministic complexity classes of languages (sets):

$$\begin{aligned}
\Phi(t) &=_{\text{def}} \{\ L \mid L \subseteq \Sigma^* \text{ and there is an algorithm } A \text{ of type } \tau \text{ accepting } L \text{ in} \\
&\qquad\qquad\qquad\qquad \Phi\text{-complexity } t\ \} \\
\Phi(O(t)) &=_{\text{def}} \bigcup_{k\geq 1} \Phi(k \cdot t) \\
\Phi(\mathrm{Pol}\ t) &=_{\text{def}} \bigcup_{k\geq 1} \Phi(t^k)
\end{aligned}$$

Proposition 1.1 *Let Φ be a complexity measure and let t, t' be resource bounds.*

1. *If $t \leq_{ae} t'$ then $\mathrm{F}\Phi(t) \subseteq \mathrm{F}\Phi(t')$.*
2. *If $t \leq_{ae} t'$ then $\Phi(t) \subseteq \Phi(t')$.*

Remarks:

1. In the definition of complexity classes, there is no restriction on a certain input alphabet but all alphabets used must be finite.
2. Numbers are encoded in dyadic. That is, for a language $L \subseteq \mathbb{N}^m$ we use the following encoding:

$$L \in \Phi(t) \iff_{\text{def}} \{\ (\mathrm{dya}(n_1), \ldots, \mathrm{dya}(n_m)) \mid (n_1, \ldots, n_m) \in L\ \} \in \Phi(t)$$

A function $f : \mathbb{N}^m \to \mathbb{N}$ can be encoded as:

$$f \in \mathrm{F\Phi}(t) \Longleftrightarrow_{\text{def}} f' \in \mathrm{F\Phi}(t) \text{ where } f' : (\{1,2\}^*)^m \to \{1,2\} \text{ is given by}$$
$$f'(x_1, \dots, x_m) =_{\text{def}} \mathrm{dya}\left(f(\mathrm{dya}^{-1}(x_1), \dots, \mathrm{dya}^{-1}(x_m))\right)$$

The dyadic encoding $\mathrm{dya} : \mathbb{N} \to \{1,2\}^*$ is recursively defined by

$$\begin{aligned} \mathrm{dya}(0) &=_{\text{def}} \varepsilon \\ \mathrm{dya}(2n+1) &=_{\text{def}} \mathrm{dya}(n)1 \\ \mathrm{dya}(2n+2) &=_{\text{def}} \mathrm{dya}(n)2 \end{aligned}$$

The decoding $\mathrm{dya}^{-1} : \{1,2\}^* \to \mathbb{N}$ is given by

$$\mathrm{dya}^{-1}(a_{n-1} \dots a_1 a_0) = \sum_{k=0}^{n-1} a_k \cdot 2^k$$

Note that the dyadic encoding is bijective (in contrast to the usual binary encoding).

1.1.2 Complexity measures for RAMs

We consider the case τ = RAM. A RAM (*random access machine*) is a model of an idealized computer based on the von Neumann architecture and consists of

- countably many register $\mathtt{R0}, \mathtt{R1}, \mathtt{R2}, \dots$, each register $\mathtt{Ri}$ containing a number $\langle\mathtt{Ri}\rangle \in \mathbb{N}$
- an instruction register $\mathtt{BR}$ containing the next instruction $\langle\mathtt{BR}\rangle$ to be executed
- a finite instruction set with instructions of several types:

type	syntax	semantics
transport	$\mathtt{R}i \leftarrow \mathtt{R}j$	$\langle\mathtt{Ri}\rangle := \langle\mathtt{Rj}\rangle,\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
	$\mathtt{RR}i \leftarrow \mathtt{R}j$	$\langle\mathtt{R}\langle\mathtt{Ri}\rangle\rangle := \langle\mathtt{Rj}\rangle,\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
	$\mathtt{R}i \leftarrow \mathtt{RR}j$	$\langle\mathtt{Ri}\rangle := \langle\mathtt{R}\langle\mathtt{Rj}\rangle\rangle,\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
arithmetic	$\mathtt{Ri} \leftarrow \mathtt{k}$	$\langle\mathtt{Ri}\rangle := k,\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
	$\mathtt{Ri} \leftarrow \mathtt{Rj} + \mathtt{Rk}$	$\langle\mathtt{Ri}\rangle := \langle\mathtt{Rj}\rangle + \langle\mathtt{Rk}\rangle,\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
	$\mathtt{Ri} \leftarrow \mathtt{Rj} - \mathtt{Rk}$	$\langle\mathtt{Ri}\rangle := \max\{\langle\mathtt{Rj}\rangle - \langle\mathtt{Rk}\rangle, 0\},\quad \langle\mathtt{BR}\rangle := \langle\mathtt{BR}\rangle + 1$
jumps	$\mathtt{GOTO\ k}$	$\langle\mathtt{BR}\rangle := k$
	$\mathtt{IF\ Ri = 0\ GOTO\ k}$	$\langle\mathtt{BR}\rangle := \begin{cases} k & \text{if } \langle\mathtt{Ri}\rangle = 0 \\ \langle\mathtt{BR} + 1\rangle & \text{otherwise} \end{cases}$
stop	$\mathtt{STOP}$	$\langle\mathtt{BR}\rangle := 0$

- a program consisting of $m \in \mathbb{N}$ instructions enumerated by $[1], [2], \dots, [m]$

The input $(x_1,\dots,x_m) \in \mathbb{N}^m$ is given by the following initial configuration:

$$\begin{aligned} \langle \mathtt{Ri} \rangle &:= x_{i+1} && \text{for } 0 \le i \le m-1 \\ \langle \mathtt{Ri} \rangle &:= 0 && \text{for } i \ge m \end{aligned}$$

A RAM computation stops when the instruction register contains zero. Then, the output is given by $\langle \mathtt{R0} \rangle$.

The complexity measures we are interested in are the following. Let β be a computation of a RAM:

$$\begin{aligned} &\text{RAM-DTIME}(\beta) =_{\text{def}} \text{number of steps (tacts, cycles) of } \beta \\ &\text{RAM-DSPACE}(\beta) =_{\text{def}} \max_{t \ge 0} \text{BIT}(\beta, t) \\ &\qquad\qquad \text{where } \text{BIT}(\beta,t) =_{\text{def}} \textstyle\sum_{i\ge 0} |\text{dya}(\langle \mathtt{Ri}\rangle_t)| + \sum_{\langle \mathtt{Ri}\rangle_t \neq 0} |\text{dya}(i)| \\ &\qquad\qquad \text{and } \langle \mathtt{Ri}\rangle_t \text{ is the content of } \mathtt{Ri} \text{ after the } t\text{-th step of } \beta \end{aligned}$$

This gives the following complexity functions for a RAM M:

$$\text{RAM-DTIME}_M(x) =_{\text{def}} \begin{cases} \text{number of steps of a computation by } M \text{ on input } x \\ \qquad\qquad\qquad\qquad \text{if } M \text{ terminates on input } x \\ \text{not defined} \qquad\qquad \text{otherwise} \end{cases}$$

$$\text{RAM-DSPACE}_M(x) =_{\text{def}} \begin{cases} \max_{t\ge 0} \text{BIT}(\text{computation by } M \text{ on input } x, t) \\ \qquad\qquad\qquad\qquad \text{if } M \text{ terminates on input } x \\ \text{not defined} \qquad\qquad \text{otherwise} \end{cases}$$

We obtain the following four complexity classes with respect to resource bounds s, t:

$$\text{FRAM-DTIME}(t), \quad \text{FRAM-DSPACE}(s), \quad \text{RAM-DTIME}(t), \quad \text{RAM-DSPACE}(s)$$

Example: We design a RAM M computing mult : $\mathbb{N} \times \mathbb{N} \to \mathbb{N} : (x,y) \mapsto x \cdot y$ in order to analyze the time complexity. The simple idea is adding y-times x. This is done by the following RAM:

```
[1]   R3 ← 1
[2]   IF R1=0 GOTO 6
[3]   R2 ← R2 + R0
[4]   R1 ← R1 - R3
[5]   GOTO 2
[6]   R0 ← R2
[7]   STOP
```

Given the input (x,y), the M takes $4 \cdot y + 4$ steps. In the worst case for inputs of size n (i.e., $x = 0$), we obtain

$$2^{n+2} \le \text{RAM-DTIME}_M(n) \le 2^{n+3} - 4.$$

1.1.3 Complexity measures for Turing machines

In the following, we adopt the reader to be familiar with the notion of a Turing machine (see, e.g., [HMU01]). According to the number of tapes Turing machines are equipped with, we consider different algorithm types τ. A Turing machines consists of:

- (possibly) a read-only input tape which either can be one-way or two-way
- k working tapes with no restrictions
- a write-only one-way output tape

The corresponding algorithm types can be taken from the following table:

τ	one working tape	k working tapes	arbitrarily many working tapes
no input tape	T	kT	multiT
one-way input tape	1-T	1-kT	1-multiT
two-way input tape	2-T	2-kT	2-multiT

The input $(x_1, \dots, x_m)$ to a Turing machines is given by the following initial configuration:

- input tape (first working tape, resp.) contains $\dots \square\square\square x_1 * \dots * x_m \square\square\square \dots$
- all other tapes are empty, i.e., they contain $\dots \square\square\square \dots$

The Turing machines stops if the halting state is taken. Then, the output is given by the configuration $\dots \square\square\square z \square \dots$ on the output tape where z is the leftmost word on the output tape which does not contain the blank symbol $\square$.

Turing machines accepting languages do not possess an output tape. Instead, they have accepting and rejecting halting states.

The complexity measures we are interested in are the following. Let τ be a Turing machine type and β be a computation of a Turing machine of type τ:

τ-DTIME$(\beta) =_{\text{def}}$ number of steps of β

τ-DSPACE$(\beta) =_{\text{def}}$ number of cells visited or containing an input symbol during computation β

This yields the following complexity functions for a Turing machine M of type τ:

$$\tau\text{-DTIME}_M(x) =_{\text{def}} \begin{cases} \text{number of steps of a computation by } M \text{ on input } x & \\ & \text{if } M \text{ terminates on input } x \\ \text{not defined} & \text{otherwise} \end{cases}$$

$$\tau\text{-DSPACE}_M(x) =_{\text{def}} \begin{cases} \text{number of cells visited or containing any input symbol} \\ \text{during a computation of } M \text{ on } x \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } M \text{ terminates on input } x \\ \text{not defined} \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

We obtain the following set of complexity classes with respect to a resource bound r:

$$\text{F}\begin{Bmatrix}0\\1\\2\end{Bmatrix}\text{-}\begin{Bmatrix}\text{T}\\k\text{T}\\\text{multiT}\end{Bmatrix}\text{-}\begin{Bmatrix}\text{DTIME}\\\text{DSPACE}\end{Bmatrix}(r), \quad \begin{Bmatrix}0\\1\\2\end{Bmatrix}\text{-}\begin{Bmatrix}\text{T}\\k\text{T}\\\text{multiT}\end{Bmatrix}\text{-}\begin{Bmatrix}\text{DTIME}\\\text{DSPACE}\end{Bmatrix}(r)$$

Example: We discuss several computational problems regarding their membership in complexity classes.

1. For the function len : $x \mapsto |x|$, it holds len $\in$ F1-T-DTIME($(1+\varepsilon)\cdot n$) for all $\varepsilon > 0$.
2. We consider two context-free languages over $\Sigma = \{0,1\}$:

$$\begin{aligned} S &=_{\text{def}} \{\ xx^R \mid x \in \{0,1\}^*\ \} \\ C &=_{\text{def}} \{\ 0^n1^n \mid n \in \mathbb{N}\ \} \end{aligned}$$

Then, the complexity classes (specified by resource bounds) S and C belong to are given in the following table (where $\varepsilon > 0$).

	T-DTIME(r)	1-T-DTIME(r)	2-T-DTIME(r)	1-T-DSPACE(r)	2-T-DSPACE(r)
S	$\varepsilon\cdot n^2$	$\varepsilon\cdot n^2$	$(1.5+\varepsilon)\cdot n$	$\varepsilon\cdot n$	$\varepsilon\cdot\log n$
C	$\varepsilon\cdot n\log n$	n	n	$\varepsilon\cdot\log n$	$\varepsilon\cdot\log n$

We want to look more closely at a result of the table: $S \in$ T-DTIME($\varepsilon\cdot n^2$) for all $\varepsilon > 0$. The basic idea of a suitable Turing machine M is to compare, in rounds, leftmost and rightmost letters of a given word and removing compared letters. Suppose we are given the word $x = 1001001001$ which belongs to S. The initial configuration is as follows (with the inital state above a cell indicating the position of the head):

$$\ldots\ \square\ \square\ \overset{s_i}{1}\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ \square\ \square\ \ldots$$

M stores the leftmost letters in two states: s_1 for letter 1 and s_0 for letter 0. The head moves to right until the first blank symbol next to the rightmost letter is reached. The head moves one cell to the left and compares that letter with the letter stored in the state. The machine stops if there is a mismatch, otherwise it moves back to left using a state s_ℓ. Then, the next round starts.

The first round is given by the following sequence of configurations:

$$\begin{array}{c}
\ldots \square\,\square\,\overset{s_i}{1}\;0\;0\;1\;0\;0\;1\;0\;0\;1\;\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\overset{s_1}{0}\;0\;1\;0\;0\;1\;0\;0\;1\;\square\,\square\,\ldots \\
\vdots \\
\ldots \square\,\square\,\square\,0\;0\;1\;0\;0\;1\;0\;0\;1\;\overset{s_1}{\square}\,\square\,\ldots \\
\ldots \square\,\square\,\square\,0\;0\;1\;0\;0\;1\;0\;0\;\overset{s'_1}{1}\;\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,0\;0\;1\;0\;0\;1\;0\;0\;\overset{s_\ell}{0}\;\square\,\square\,\square\,\ldots \\
\vdots \\
\ldots \square\,\square\,\overset{s_\ell}{\square}\,0\;0\;1\;0\;0\;1\;0\;0\;\square\,\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\overset{s_i}{0}\;0\;1\;0\;0\;1\;0\;0\;\square\,\square\,\square\,\ldots
\end{array}$$

Note that this round takes $2|x| + 1$ steps.

The second round is given by the following sequence of configurations:

$$\begin{array}{c}
\ldots \square\,\square\,\square\,\overset{s_i}{0}\;0\;1\;0\;0\;1\;0\;0\;\square\,\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\square\,\overset{s_0}{0}\;1\;0\;0\;1\;0\;0\;\square\,\square\,\square\,\ldots \\
\vdots \\
\ldots \square\,\square\,\square\,\square\,0\;1\;0\;0\;1\;0\;0\;\overset{s_0}{\square}\,\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\square\,0\;1\;0\;0\;1\;0\;\overset{s'_0}{0}\;\square\,\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\square\,0\;1\;0\;0\;1\;\overset{s_\ell}{0}\;\square\,\square\,\square\,\square\,\ldots \\
\vdots \\
\ldots \square\,\square\,\square\,\overset{s_\ell}{\square}\,0\;1\;0\;0\;1\;0\;\square\,\square\,\square\,\square\,\ldots \\
\ldots \square\,\square\,\square\,\square\,\overset{s_i}{0}\;1\;0\;0\;1\;0\;\square\,\square\,\square\,\square\,\ldots
\end{array}$$

Note that this round takes $2(|x| - 2) + 1$ steps.

After a number of rounds all letters are removed. The machine accepts if and only if there are no mismatches and in the last round there are exactly two letters on the tape.

The time complexity for x such that $|x| = 2m$ can be estimated as follows:

$$\text{T-DTIME}_M(x) \leq \sum_{i=0}^{m-1} 4m - 4i + 1 = 4m^2 - \frac{4m(m-1)}{2} + m = 2m^2 + 3m \leq_{\text{ae}} \frac{2}{3} \cdot |x|^2$$

Now, instead of comparing one symbol, compare k symbols (by increasing the numbers of states exponentially). This gives the following complexity analysis for x:

$$\begin{aligned}
\text{T-DTIME}_{M'}(x) &\leq \sum_{i=0}^{\lfloor m/k \rfloor} (|x| - 2ki) + (|x| - 2ki - k) + 1 \quad + k \\
&= \sum_{i=0}^{\lfloor m/k \rfloor} 4m - (4i+1)k + 1 \quad + k \\
&\leq (4m+1)\left(\frac{m}{k} + 1\right) \quad + k \\
&\leq_{\text{ae}} \frac{2}{k} \cdot |x|^2
\end{aligned}$$

For $k \geq 2 \cdot \varepsilon^{-1}$, we obtain $S \in \text{T-DTIME}(\varepsilon \cdot n^2)$.

1.2 Nondeterministic measures

Nondeterminism of algorithms is a certain kind of parallelism:

- possibly many instructions to perform next given a situation
- realized in parallel (by identical copies of the machine/algorithm)
- number of instructions limited by a constant number

For a nondeterministic RAM, this means that instructions may have same numbers. For a nondeterministic Turing machine, this means that, given a situation (i.e., a state and symbols on tapes), there are many transitions applicable.

Nondeterministic machines produce computation trees. Let k be the maximum number of nondeterministic branches of M. A *computation path* of M on input x is a word $a_1 \ldots a_r \in \{1, \ldots, k\}^*$ such that a_t is the a_t-th instruction of the same number performed in the t-th step of the computation. Note that not each word from $\{1, \ldots, k\}^*$ describes a computation path.

As an example consider the computation tree above. Here, x is the input to the algorithm. The red computation path can be described by the word $1221232 \in \{1, 2, 3\}^*$. In contrast, the word $1212 \in \{1, 2, 3\}$ does not correspond to a computation path on input x.

As it is not obvious how to define nondeterministic function classes, we focus on decision problems only in the forthcoming. We define the notion of "acceptance by a nondeterministic machine:"

- τ is an algorithm type
- A is a nondeterministic algorithm of type τ
- x is an input
- z is a computation path of A on x

Then, we define:

- $\varphi_A(x|z) =_{\text{def}}$ result of A on x along z
- A accepts $x \Longleftrightarrow_{\text{def}}$ there is a z of A on x such that $\varphi_A(x|z) = 1$ (i.e., there is an accepting computation path of A on x)
- A accepts $L \subseteq \Sigma^* \Longleftrightarrow_{\text{def}} L = \{ x \in \Sigma^* \mid A \text{ accepts } x \}$

Note that deterministic algorithms are a subclass of nondeterministic algorithms.

A *complexity measure* for nondeterministic algorithm A of type τ is a mapping Φ:

$$\Phi \ : \ \text{computation path of } A \text{ of } \tau \text{ on } x \ \mapsto \ r \in \mathbb{N}$$

Example: Standard complexity measures are the following:

$$\Phi = \tau\text{-NTIME} \qquad \Phi = \tau\text{-NSPACE}$$

Here, "N" indicates that algorithms are nondeterministic. Note that if A is actually deterministic then the following holds:

$$\begin{aligned} \tau\text{-NTIME(computation path)} &= \tau\text{-DTIME(computation)} \\ \tau\text{-NSPACE(computation path)} &= \tau\text{-DSPACE(computation)} \end{aligned}$$

We define the following complexity functions: Let Φ be a complexity measure for nondeterministic algorithms A of type τ.

$\Phi_A(x|z) =_{\text{def}} \Phi(\text{computation path } z \text{ of } A \text{ on } x)$

$\Phi_A(x) =_{\text{def}} \min \{ \ \Phi_A(x|z) \mid \varphi_A(x|z) = 1 \ \}$ (we set $\min \emptyset =_{\text{def}} 0$!)

$\Phi_A(n) =_{\text{def}} \max\limits_{|x|=n} \Phi_A(x)$

We say that a nondeterministic algorithm A accepts a language L in Φ-complexity t if and only if A accepts L and $\Phi_A \leq_{\text{ae}} t$.

We obtain the following nondeterministic complexity classes of languages:

$$\Phi(t) =_{\text{def}} \{\ L \mid L \subseteq \Sigma^* \text{ and there is a nondeterministic algorithm } A \text{ of type } \tau \text{ accepting } L \text{ in } \Phi\text{-complexity } t\ \}$$

$$\Phi(O(t)) =_{\text{def}} \bigcup_{k\geq 1} \Phi(k \cdot t)$$

$$\Phi(\text{Pol } t) =_{\text{def}} \bigcup_{k\geq 1} \Phi(t^k)$$

Proposition 1.2 *Let τ be any algorithm type, and let tt' be resource bounds.*

1. *If $t \leq_{\text{ae}} t'$ then $\Phi(t) \subseteq \Phi(t')$ for all Φ.*
2. *τ-DTIME$(t) \subseteq \tau$-NTIME(t).*
3. *τ-DSPACE$(t) \subseteq \tau$-NSPACE(t).*

For deterministic complexity classes, the following equivalences are true:

$$\begin{aligned} L \in \tau\text{-DTIME}(t) &\iff \overline{L} \in \tau\text{-DTIME}(t) \\ L \in \tau\text{-DSPACE}(t) &\iff \overline{L} \in \tau\text{-DSPACE}(t) \end{aligned}$$

The same statements need not be true for τ-NTIME and τ-NSPACE because of the non-symmetrical acceptance conditions.

Examples: We discuss the nondeterministic complexities of the problems S and C with respect to various computational models. According to the remarks above we make a distinction between S, $\overline{S}$, C, and $\overline{C}$. The differences in the complexities illustrate the remark once more.

	T-NTIME(r)	1-T-NTIME(r)	2-T-NTIME(r)	1-T-NSPACE(r)	2-T-NSPACE(r)
S	$\varepsilon \cdot n^2$	n	n	$\varepsilon \cdot n$	$\varepsilon \cdot \log n$
$\overline{S}$	$\varepsilon \cdot n \cdot \log n$	n	n	$\varepsilon \cdot \log n$	$\varepsilon \cdot \log n$
C	$\varepsilon \cdot n \cdot \log n$	n	n	$\varepsilon \cdot \log n$	$\varepsilon \cdot \log n$
$\overline{C}$	$\varepsilon \cdot n \cdot \log\log n$	n	n	$\varepsilon \cdot \log\log n$	$\varepsilon \cdot \log\log n$

We discuss two particular cases in more detail:

1. In order to show that $\overline{S} \in$ T-NTIME$(n \cdot \log n)$, define M to be the T-TM that, on input x,
   (a) guesses a position i,

(b) determines the letter at the i-th position from left (i.e., x_i),

(c) determines the letter at the i-th position from right (i.e., x_{n-i}),

(d) accepts if and only if the letters are different and $|x|$ is even.

Clearly, M accepts $\overline{S}$. Analyzing the complexity is as follows: if $x \in \overline{S}$ then there is a position $0 \leq i \leq \frac{n}{2}$ such that $x_i \neq x_{n-i}$. Let z be the computation path for guessing i. Then, we obtain:

$$\text{T-NTIME}_M(x|z) \leq_{\text{a.e.}} |x| \cdot \log |x|$$

Thus, $\overline{S}$ belongs to T-NTIME$(n \cdot \log n)$.

2. In order to show $\overline{C} \in$ T-NTIME$(n \cdot \log n)$, define M to be the T-TM that, on input x,

   (a) guesses a $k \in \mathbb{N} \setminus \{0, 1\}$,

   (b) determines $\alpha_k =_{\text{def}} \text{mod}(|x|_0, k)$ ($|x|_0$ denotes the number of 0's in x),

   (c) determines $\beta_k =_{\text{def}} \text{mod}(|x|_1, k)$ ($|x|_1$ denotes the number of 1's in x),

   (d) accepts if and only if $\alpha_k \neq \beta_k$ or there occurs a 1 followed by 0.

   Clearly, M accepts $\overline{C}$. We analyze the complexity of M: if $x \notin \overline{C}$ (i.e., $x = 0^n 1^n$) then there is no accepting path; if $x \in \overline{C}$ (i.e., $|x|_0 \neq |x|_1$) then there is an accepting path z_k for $k \leq n$. Thus, we obtain

$$\text{T-NTIME}_M(x|z_k) \leq |x| \cdot \log k$$

   But this only proves an $n \cdot \log n$-bound. We can do a finer analysis: The Chinese remainder theorem states that $\mathbb{Z}_{p \cdot q} \cong (\mathbb{Z}_p) \times (\mathbb{Z}_q)$ for different prime numbers p and q. That is, if $x \in \overline{C}$, we find $\alpha_k \neq \beta_k$ for at least one of the first m prime numbers $p_1, \ldots, p_m$ such that $n \leq p_1 \cdot \cdots \cdot p_m$. So, $m =_{\text{def}} \lceil \log n \rceil$ is enough (since $n \leq 2^{\log n} \leq 2^m \leq p_1 \cdot \cdots \cdot p_m$). By the prime number theorem we know that $p_m \leq c \cdot m \cdot \log m$ for some $c > 1$. Then, we conclude

$$\begin{aligned}
\text{T-NTIME}_M(x|z_k) &\leq |x| \cdot \log k \\
&\leq |x| \cdot \log p_m \\
&\leq |x| \cdot \log(c \cdot \log |x| \cdot \log\log |x|)) \\
&\leq 2|x| \cdot \log\log |x| + |x| \cdot \log c \ \leq_{a.e} \ 3|x| \cdot \log\log |x|
\end{aligned}$$

   Using linear compression we obtain $\overline{C} \in$ T-NTIME$(n \cdot \log\log n)$.

1.3 Type-independent measures

We determine complexity classes τ-DSPACE(s) and τ-NSPACE(s) that do not depend on τ for $s(n) \geq n$.

Theorem 1.3 *For* $\mathrm{X} \in \{\mathrm{D}, \mathrm{N}\}$ *and all* $s(n) \geq 0$, *the following holds:*

1. i-kT-XSPACE(s) = i-multiT-XSPACE(s) *for* $i \in \{0,1,2\}$ *and* $k \geq 1$
2. T-XSPACE(s) = 1-T-XSPACE(s) = 2-T-XSPACE(s) *for* $s(n) \geq n$
3. 1-T-XSPACE$(s) \subseteq$ 2-T-XSPACE(s)
4. RAM-XSPACE$(O(s))$ = T-XSPACE(s) *for* $s(n) \geq n$

Turing machines equipped with a 2-way input tape are most flexible in sublinear space. We define the following machine-independent space-complexity classes (for $s(n) \geq 0$):

$$\begin{aligned} \mathrm{DSPACE}(s) &=_{\mathrm{def}} \text{2-T-DSPACE}(s) \\ \mathrm{NSPACE}(s) &=_{\mathrm{def}} \text{2-T-NSPACE}(s) \end{aligned}$$

Proposition 1.4 DSPACE$(s) \subseteq$ NSPACE(s) *for all* $s(n) \geq 0$.

We introduce names for special space-complexity classes:

$$\begin{aligned} \mathrm{L} &=_{\mathrm{def}} \mathrm{DSPACE}(\log n) \\ \mathrm{NL} &=_{\mathrm{def}} \mathrm{NSPACE}(\log n) \\ \mathrm{LIN} &=_{\mathrm{def}} \mathrm{DSPACE}(O(n)) \\ \mathrm{NLIN} &=_{\mathrm{def}} \mathrm{NSPACE}(O(n)) \\ \mathrm{PSPACE} &=_{\mathrm{def}} \mathrm{DSPACE}(\mathrm{Pol}\ n) \\ \mathrm{NPSPACE} &=_{\mathrm{def}} \mathrm{NSPACE}(\mathrm{Pol}\ n) \end{aligned}$$

Proposition 1.5 *The following inclusions are true: ...*

Remarks:

1. For arbitrary $s : \mathbb{N} \to \mathbb{N}$, it is open whether DSPACE$(s) \subset$ NSPACE(s) or whether DSPACE(s) = NSPACE(s).
2. Special open questions are: $\mathrm{L} \overset{?}{=} \mathrm{NL}$, $\mathrm{LIN} \overset{?}{=} \mathrm{NLIN}$ (aka the first LBA problem)

Theorem 1.6 *For* X $\in \{\mathrm{D}, \mathrm{N}\}$ *and all* $t(n) \geq n$*, the following holds:*

1. i-kT-XTIME(Pol t) = i-multiT-XTIME(Pol t) *for* $i \in \{0, 1, 2\}$ *and* $k \geq 1$
2. T-XTIME(Pol t) = 1-T-XTIME(Pol t) = 2-T-XTIME(Pol t)
3. RAM-XTIME(Pol t) = T-XTIME(Pol t)

Remarks: There are some results regarding tight simulations for "easy-to-compute" time bounds $t(n) \geq n$:

- multiT-DTIME(t) $\subseteq$ T-DTIME(t^2)
- multiT-DTIME(t) $\subseteq$ 2T-DTIME($t \cdot \log t$)
- multiT-NTIME(t) = 2T-NTIME(t)
- multiT-DTIME(t) $\subseteq$ RAM-DTIME $(O(t/\log t))$ for $t(n) \geq n \cdot \log n$
- RAM-DTIME(t) $\subseteq$ multiT-DTIME(t^3)
- multiT-NTIME(t) $\subseteq$ RAM-NTIME($O(t)$) $\subseteq$ multiT-NTIME(t^3)

We define the following machine-independent time-complexity classes (for $t(n) \geq n$):

$$\begin{aligned} \mathrm{DTIME}(\mathrm{Pol}\ t) &=_{\mathrm{def}} \mathrm{T\text{-}DTIME}(\mathrm{Pol}\ t) \\ \mathrm{NTIME}(\mathrm{Pol}\ t) &=_{\mathrm{def}} \mathrm{T\text{-}NTIME}(\mathrm{Pol}\ t) \end{aligned}$$

Proposition 1.7 DTIME(Pol t) $\subseteq$ NTIME(Pol t) *for all* $t(n) \geq n$.

We introduce name for special time-complexity classes:

$$\begin{aligned} \mathrm{P} &=_{\mathrm{def}} \mathrm{DTIME}(\mathrm{Pol}\ n) \\ \mathrm{NP} &=_{\mathrm{def}} \mathrm{NTIME}(\mathrm{Pol}\ n) \\ \mathrm{E} &=_{\mathrm{def}} \mathrm{DTIME}(\mathrm{Pol}\ 2^n) \\ \mathrm{NE} &=_{\mathrm{def}} \mathrm{NTIME}(\mathrm{Pol}\ 2^n) \\ \mathrm{EXP} &=_{\mathrm{def}} \mathrm{DTIME}\left(2^{\mathrm{Pol}\ n}\right) \\ \mathrm{NEXP} &=_{\mathrm{def}} \mathrm{NTIME}\left(2^{\mathrm{Pol}\ n}\right) \end{aligned}$$

Proposition 1.8 *The following inclusions are true: ...*

Remarks:

1. For arbitrary $t : \mathbb{N} \to \mathbb{N}$, it is open whether DTIME(t) $\subset$ NTIME(t) or whether DTIME(t) $=$ NTIME(t); an exception is the following proven result: DTIME($O(n)$) $\subset$ NTIME($O(n)$).

2. Special open questions are P $\overset{?}{=}$ NP, E $\overset{?}{=}$ NE, and EXP $\overset{?}{=}$ NEXP

Complexity hierarchies 2

Let Φ be any complexity measure. Which of the following both (complementary) statements is true?

1. Is there a computable function $t : \mathbb{N} \to \mathbb{N}$ such that $\Phi(t)$ contains all decidable sets?
2. Given any computable function $t : \mathbb{N} \to \mathbb{N}$, is there a decidable set A such that $A \notin \Phi(t)$?

In the case that the answer to the second question is "yes":

3. Given t, how much greater has t' to be in order to get $\Phi(t) \subset \Phi(t')$?

That is: Is there an infinite hierarchy of complexity classes $\Phi(t)$?

2.1 Deterministic space

We consider DSPACE = 2-T-DSPACE.

As a preliminary remark we syntactically define a 2-T-TM M as a tuple $(\Sigma, \Delta, S, \delta, s_+, s_-)$ where

- Σ is a finite input alphabet
- Δ is a finite internal alphabet
- S is a finite set of states
- $\delta : S \times \Sigma \times \Delta \to S \times \Delta \times \{L, 0, R\}^2$
- s_+ is the accepting halting state
- s_- is the rejecting halting state

Let $|M|$ denote the description length of M:

$$|M| \geq \|S\| \cdot \|\Sigma\| \cdot \|\Delta\|$$

Lemma 2.1 *If a deterministic* 2-T-*TM M terminates on input x then it holds*

$$\text{2-T-DTIME}_M(x) \leq \|x\| \cdot 2^{|M| \cdot \text{DSPACE}_M(x)}$$

Corollary 2.2 $\mathrm{DSPACE}(s) \subseteq \mathrm{DTIME}(2^{O(s)})$ *for* $s(n) \geq \log n$.

Theorem 2.3 *For every computable function* $s : \mathbb{N} \to \mathbb{N}$ *there is a decidable language* L *such that* $L \notin \mathrm{DSPACE}(s)$.

A function $s : \mathbb{N} \to \mathbb{N}$ is said to be *space-constructible* if and only if there is a 2-T-TM M such that $\mathrm{DSPACE}_M(x) = s(|x|)$.

Proposition 2.4 *Let* $s : \mathbb{N} \to \mathbb{N}$ *be a function. Then, the following equivalence holds:*

$$s \text{ is space-constructible} \iff s \circ \mathrm{len} \in \mathrm{FDSPACE}(s)$$

Remarks:

1. $\mathrm{len} : \Sigma^* \to \mathbb{N} : x \mapsto |x|$
2. $s \circ \mathrm{len} : \Sigma^* \to \mathbb{N} : x \mapsto s(\mathrm{len}(x)) = s(|x|)$

Examples: n^k for $k \geq 1$, $\log n$, 2^n are space-constructible functions.

Proposition 2.5 *Let* s *and* s' *be space-constructible functions. Then,*

1. $s + s'$, $s \cdot s'$, $\max(s, s')$ *are space-constructible,*
2. $s \circ s'$ *is space-constructible, if* $s(n) \geq n$.

Theorem 2.6 (Hierarchy Theorem) *Let* s *and* s' *be space-constructible functions,* $s(n) \geq \log n$. *If* $s' = o(s)$ *then* $\mathrm{DSPACE}(s') \subset \mathrm{DSPACE}(s)$.

Theorem 2.7 (Linear Compression) *For every function* $s : \mathbb{N} \to \mathbb{N}$,

$$\mathrm{DSPACE}(s) = \mathrm{DSPACE}(O(s)).$$

Remarks:

1. Gap Theorem (Trakhtenbrot 1964, Borodin 1972): For every computable $r : \mathbb{N} \to \mathbb{N}$ there is a computable $s : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{DSPACE}(s) = \mathrm{DSPACE}(r \circ s)$. (So, s is not space-constructible.)
2. There is an s such that $\mathrm{DSPACE}(s) = \mathrm{DSPACE}(2^s)$.
3. $\mathrm{REG} = \mathrm{DSPACE}(1) = \mathrm{DSPACE}(s)$ for $s = o(\log \log n)$.

2.2 Nondeterministic space

We consider NSPACE = 2-T-NSPACE.

Theorem 2.8 (Hierarchy Theorem) *For every space-constructible $s(n) \geq \log n$ and every s' such that $s'(n+1) = o(s(n))$,*

$$\text{NSPACE}(s') \subset \text{NSPACE}(s).$$

Remarks:

1. Diagonalization does not work for nondeterministic measures.
2. Proof is based on *recursive padding*. The padding technique is as follows: For a set A and a function $r(n) > n$ define

   $$A_r =_{\text{def}} \{\ xba^{r(|x|)-|x|-1} \mid b \neq a \wedge x \in A\ \}.$$

   Then, one can prove that

   $$A_r \in \text{NSPACE}(s) \iff A \in \text{NSPACE}(s \circ r)$$

3. *Upward translation of equality*: We show that

   $$\text{NSPACE}(n^2) \subseteq \text{NSPACE}(n) \implies \text{NSPACE}(n^3) \subseteq \text{NSPACE}(n).$$

   Let $r(n) =_{\text{def}} n^{3/2}$ and note that $n^3 = (n^{3/2})^2$. Then, conclude as follows:

   $$\begin{aligned} A \in \text{NSPACE}(n^3) &\implies A_r \in \text{NSPACE}(n^2) \\ &\implies A_r \in \text{NSPACE}(n) \\ &\implies A \in \text{NSPACE}(n^{3/2}) \\ &\implies A \in \text{NSPACE}(n^2) \\ &\implies A \in \text{NSPACE}(n) \end{aligned}$$

4. Savitch's Theorem: $\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2)$. We obtain that

   $$\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2) \subset \text{DSPACE}(s^3) \subseteq \text{NSPACE}(s^3).$$

   In particular, $\text{NSPACE}(n) \subset \text{NSPACE}(n^2)$

Theorem 2.9 (Linear Compression) *For every function $s : \mathbb{N} \to \mathbb{N}$,*

$$\text{NSPACE}(s) = \text{NSPACE}(O(s)).$$

2.3 Deterministic time

Theorem 2.10 (Hierarchy Theorem) *For "easy-to-compute" functions $t(n) \geq n$ and for all $t'(n) = o(t(n))$,*

$$\text{multiT-DTIME}(t') \subset \text{multiT-DTIME}(t \cdot \log t).$$

Remarks:

1. Proof is by diagonalization; the $\log t$ factor appears because of the simulation of arbitrarily many tapes by a fixed number of tapes.
2. Using recursive padding: multiT-DTIME(t') $\subset$ multiT-DTIME($t \cdot \sqrt{\log t}$)
3. For $k \geq 2$ fixed: kT-DTIME(t') $\subset$ kT-DTIME(t)
4. RAM-DTIME(t) $\subset$ RAM-DTIME($c \cdot t$) for some $c > 1$.

Theorem 2.11 (Linear Speed-up) *For $t(n) \geq c \cdot n$ such that $c > 1$,*

$$\text{multiT-DTIME}(t) = \text{multiT-DTIME}(O(t)).$$

Theorem 2.12 multiT-DTIME(n) $\subset$ multiT-DTIME($O(n)$).

2.4 Nondeterministic time

Theorem 2.13 (Hierarchy Theorem) *For "easy-to-compute" functions $t(n) \geq n$ and for all $t'(n) \geq n$ such that $t'(n+1) = o(t(n))$,*

$$\text{multiT-NTIME}(t') \subset \text{multiT-NTIME}(t)$$

Theorem 2.14 (Linear Speed-up) *For $t(n) \geq n$,*

$$\text{multiT-NTIME}(t) = \text{multiT-NTIME}(O(t)).$$

Remarks:

1. The linear speed-up already holds for $t(n) = n$ in contrast to the linear speed-up for deterministic time.
2. multiT-DTIME(n) $\subset$ multiT-NTIME(n).
3. multiT-DTIME($O(n)$) $\subset$ multiT-NTIME($O(n)$).

Relations between space and time complexity 3

In this chapter we relate DSPACE, NSPACE, DTIME, and NTIME.

3.1 Space versus time

We first study time-efficient simulations of space-bounded computations.

Proposition 3.1 *If a nondeterministic* 2-T-*TM M accepts a language L in space $s(n) \geq \log n$ then M accepts L in time $2^{O(s)}$.*

Theorem 3.2 *Let $s(n) \geq \log n$ be space-constructible. Then,*

$$\mathrm{NSPACE}(s) \subseteq \mathrm{DTIME}(2^{O(s)}).$$

Corollary 3.3

1. NL $\subseteq$ P.
2. NLIN $\subseteq$ E.
3. NPSPACE $\subseteq$ EXP.

We turn to space-efficient simulations of time-bounded computations. Clearly, it holds that DTIME$(t) \subseteq$ DSPACE(t) and NTIME$(t) \subseteq$ NSPACE(t).

Theorem 3.4 *For any space-constructible function $t(n) \geq n$,*

$$\text{T-NTIME}(t) \subseteq \mathrm{DSPACE}(t).$$

Corollary 3.5

1. -NTIME(Pol t) $\subseteq$ -DSPACE(Pol t) *for any space-constructible function $t(n) \geq n$.*
2. NP $\subseteq$ PSPACE.

3.2 Nondeterministic space versus deterministic space

Lemma 3.6 *For all space-constructible functions $s(n), t(n) \geq \log n$, the following holds: If a language L is accepted by a* 2-T-*NTM in space s and time 2^t then L is accepted by a* 2-T-*DTM in space $s \cdot t$.*

Theorem 3.7 (Savitch 1970) *For any space-constructible function $s(n) \geq \log n$,*

$$\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2).$$

Corollary 3.8

1. $\text{NL} \subseteq \text{DSPACE}(\log^2 n)$.
2. $\text{NLIN} \subseteq \text{DSPACE}(n^2)$.
3. $\text{NPSPACE} = \text{PSPACE}$.

3.3 Complementing nondeterministic space

For each set class $\mathcal{K}$ define

$$\text{co}\mathcal{K} =_{\text{def}} \{\, \overline{A} \mid A \in \mathcal{K} \,\}.$$

Note that in order to prove $\mathcal{K} = \text{co}\mathcal{K}$ it is enough to prove one of the inclusions $\mathcal{K} \subseteq \text{co}\mathcal{K}$ or $\text{co}\mathcal{K} \subseteq \mathcal{K}$.

Theorem 3.9 (Szelepczényi, Immerman 1987) *For any space-constructible functions $s(n) \geq \log n$,*

$$\text{coNSPACE}(s) = \text{NSPACE}(s).$$

Corollary 3.10

1. $\text{coNL} = \text{NL}$.
2. $\text{coNLIN} = \text{NLIN}$.

3.4 Open problems in complexity theory

The following table shows that general complexity-theoretic open problems. The most important open problems are framed.

General Problem	Special cases
$\mathrm{DSPACE}(s) \overset{?}{=} \mathrm{NSPACE}(s)$	$\boxed{\mathrm{L} \overset{?}{=} \mathrm{NL}}$
	$\boxed{\mathrm{LIN} \overset{?}{=} \mathrm{NLIN}}$
$\mathrm{NSPACE}(s) \overset{?}{=} \mathrm{DTIME}(2^{O(s)})$	$\boxed{\mathrm{NL} \overset{?}{=} \mathrm{P}}$
	$\mathrm{NLIN} \overset{?}{=} \mathrm{E}$
	$\mathrm{PSPACE} \overset{?}{=} \mathrm{EXP}$
$\mathrm{DTIME}(\mathrm{Pol}\ t) \overset{?}{=} \mathrm{NTIME}(\mathrm{Pol}\ t)$	$\boxed{\mathrm{P} \overset{?}{=} \mathrm{NP}}$
	$\mathrm{E} \overset{?}{=} \mathrm{NE}$
	$\mathrm{EXP} \overset{?}{=} \mathrm{NEXP}$
$\mathrm{NTIME}(\mathrm{Pol}\ t) \overset{?}{=} \mathrm{DSPACE}(\mathrm{Pol}\ t)$	$\boxed{\mathrm{NP} \overset{?}{=} \mathrm{PSPACE}}$
	$\mathrm{NE} \overset{?}{=} \mathrm{DSPACE}(2^{O(s)})$
	$\mathrm{NEXP} \overset{?}{=} \mathrm{DSPACE}(2^{\mathrm{Pol}\ s})$
$\mathrm{NTIME}(\mathrm{Pol}\ t) \overset{?}{=} \mathrm{coNTIME}(\mathrm{Pol}\ t)$	$\boxed{\mathrm{NP} \overset{?}{=} \mathrm{coNP}}$
	$\mathrm{NE} \overset{?}{=} \mathrm{coNE}$
	$\mathrm{NEXP} \overset{?}{=} \mathrm{coNEXP}$

Remark: The classes $\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}, \mathrm{coNP}, \mathrm{PSPACE}, \ldots$ are the most important complexity classes as they contain many practically relevant problems. Furthermore, they are reference classes for more advanced or refined computational models, e.g., quantum, randomized, or parallel computations.

In the following we want to study relations among open problems (*upward translation of equality*): For a function $r : \mathbb{N} \to \mathbb{N}$, a set $A \subseteq \Sigma^*$, and a fixed, distinct letters $a, b \in \Sigma$ define the set

$$A_r =_{\mathrm{def}} \left\{ xba^{r(|x|)-|x|-1} \mid b \neq a \wedge x \in A \right\}$$

A function $t : \mathbb{N} \to \mathbb{N}$ is *time-constructible* if and only if $t \circ \text{len} \in \text{FDTIME}(\text{Pol } t)$, i.e., $x \mapsto t(|x|)$ is computable in time $t(|x|)^k$ for some $k \geq 1$.

Lemma 3.11 (Padding Lemma) *Let $X \in \{\text{D}, \text{N}\}$.*

1. *For any space-constructible function $s(n) \geq \log n$ and any function $r(n) > n$ such that $r \circ \text{len} \in \text{DSPACE}(s \circ r)$,*

$$A \in \text{XSPACE}(s \circ r) \iff A_r \in \text{XSPACE}(s).$$

2. *For any time-constructible function $t(n) \geq n$ and any function $r(n) > n$,*

$$A \in \text{XTIME}(\text{Pol } t \circ r) \iff A_r \in \text{XTIME}(\text{Pol } t).$$

Theorem 3.12 *Let s be a space-constructible function, and let t be time-constructible function.*

1. $\text{L} = \text{NL} \implies \text{DSPACE}(s) = \text{NSPACE}(s)$ *for* $s(n) > \log n$
2. $\text{LIN} = \text{NLIN} \implies \text{DSPACE}(s) = \text{NSPACE}(s)$ *for* $s(n) > n$
3. $\text{NL} = \text{P} \implies \text{NSPACE}(s) = \text{DTIME}(2^{O(s)})$ *for* $s(n) > \log n$
4. $\text{P} = \text{NP} \implies \text{DTIME}(\text{Pol } t) = \text{DSPACE}(\text{Pol } t)$ *for* $t(n) > n$
5. $\text{NP} = \text{PSPACE} \implies \text{NTIME}(\text{Pol } t) = \text{DSPACE}(\text{Pol } t)$ *for* $t(n) > n$
6. $\text{NP} = \text{coNP} \implies \text{NTIME}(\text{Pol } t) = \text{coNTIME}(\text{Pol } t)$ *for* $t(n) > n$

Corollary 3.13

1. $\text{L} = \text{NL} \implies \text{LIN} = \text{NLIN}$
2. $\text{NL} = \text{P} \implies \text{NLIN} = \text{E} \implies \text{PSPACE} = \text{EXP}$
3. $\text{P} = \text{NP} \implies \text{E} = \text{NE} \implies \text{EXP} = \text{NEXP}$
4. $\text{NP} = \text{PSPACE} \implies \text{NE} = \text{DSPACE}(2^{O(n)}) \implies \text{NEXP} = \text{DSPACE}(2^{\text{Pol } n})$

Lower bounds 4

We turn to a complexity theory for computational problems.

Let $t : \mathbb{N} \to \mathbb{N}$ be a resource bound, and let A be any language. Then,

- t is a *lower bound* for A with respect to Φ-complexity if and only if $A \notin \Phi(t)$,
- t is an *upper bound* for A with respect to Φ-complexity if and only if $A \in \Phi(t)$.

Remark: If Φ admits linear compression or speed-up then there is no greatest lower bound. Suppose $A \notin \Phi(t)$ and t is greatest lower bound for A. Then, $A \notin \Phi(t)$ but $A \in \Phi(2t)$. By linear compression/speed-up, $A \in \Phi(t)$ which is a contradiction.

The goal in this chaper is to provide techniques for proving lower bound for concrete problems. We consider two techniques: the *completeness method* (based on diagonalization) and the *counting method* (based on the pigeonhole principle).

4.1 The completeness method

Instead of proving explicite resource bound, we compare computational problems, i.e., we prove statements like "A is computationally not harder then B."

4.1.1 Reducibilities

Definition 4.1 *Let $A \subseteq \Sigma^*$ and $B \subseteq \Delta^*$ be languages.*

1. *$A \leq_m^p B$ if and only if there is an $f \in \mathrm{FP}$ such that for all $x \in \Sigma^*$*
$$x \in A \iff f(x) \in B.$$
2. *$A \leq_m^{\log} B$ if and only if there is an $f \in \mathrm{FL}$ such that for all $x \in \Sigma^*$*
$$x \in A \iff f(x) \in B.$$
3. *$A \leq_m^{\log} B$ if and only if there exist an $f \in \mathrm{FL}$ and a $c > 0$ such that for all $x \in \Sigma^*$, $|f(x)| \leq c \cdot |x|$ and*
$$x \in A \iff f(x) \in B.$$

Example: Consider the following two sets:

$$\begin{aligned} \textsc{Subset Sum} &=_{\text{def}} \left\{ (a_1,\dots,a_m,b) \mid (\exists I \subseteq \{1,\dots,m\}) \left[\textstyle\sum_{i\in I} a_i = b\ \right]\right\} \\ \textsc{Partition} &=_{\text{def}} \left\{ (a_1,\dots,a_m) \mid (\exists I \subseteq \{1,\dots,m\}) \left[\textstyle\sum_{i\in I} a_i = \sum_{i\notin I} a_i\right]\ \right\} \end{aligned}$$

We show Subset Sum $\leq_m^{\log}$ Partition. Consider the following function

$$f : (a_1,\dots,a_m,b) \mapsto (a_1,\dots,a_m,b+1,N-b+1)$$

where $N =_{\text{def}} \sum_{i=1}^m a_i$. That is, $f(a_1,\dots,a_m,b) = (a'_1,\dots,a'_{m+2})$ such that $a'_i = a_i$ for $i \in \{1,\dots,m\}$, $a'_{m+1} = b$, and $a'_{m+2} = N-b$.

Claim: $(a_1,\dots,a_m,b) \in$ Subset Sum $\iff f(a_1,\dots,a_m,b) \in$ Partition.

For $(\Rightarrow)$ let $(a_1,\dots,a_m,b) \in$ Subset Sum, i.e., there is an $I \subseteq \{1,\dots,m\}$ such that $\sum_{i\in I} a_i = b$. Define $I' =_{\text{def}} I \cup \{m+2\}$. Then,

$$\begin{aligned} \sum_{i\in I'} a'_i &= \sum_{i\in I} a'_i + a'_{m+2} = \sum_{i\in I} a_i + N - b + 1 = b + N - b + 1 = N+1 \\ \sum_{i\notin I'} a'_i &= \sum_{i\notin I} a'_i + a'_{m+1} = \sum_{i\notin I} a_i + b + 1 = N - b + b + 1 = N+1 \end{aligned}$$

Hence, $f(a_1,\dots,a_m,b) \in$ Partition.

For $(\Leftarrow)$ let $f(a_1,\dots,a_m,b) \in$ Partition, ie., there is an $I' \subseteq \{1,\dots,m+2\}$ such that

$$\sum_{i\in I'} a'_i = \sum_{i\notin I'} a'_i = \frac{1}{2}\cdot\sum_{i=1}^{m+2} a'_i = N+1.$$

It holds that $m+2 \in I'$ if and only if $m+1 \notin I'$ (since $a'_{m+1} + a'_{m+2} = N+2$). Without loss of generality, assume that $m+2 \in I'$. Define $I =_{\text{def}} I' \setminus \{m+2\}$. Then,

$$N+1 = \sum_{i\in I'} a'_i = \sum_{i\in I} a'_i + a'_{m+2} = \sum_{i\in I} a_i + N - b + 1$$

Thus, $\sum_{i\in I'} a_i = b$. Hence, $(a_1,\dots,a_m,b) \in$ Subset Sum.

The running space of an appropriate Turing machine summing up all a_i's to compute N is certainly $O(\log n)$. That is, $f \in \mathrm{FL}$.

Proposition 4.2 *Let A and B be any languages.*

1. $A \leq_m^{\log\text{-lin}} B \implies A \leq_m^{\log} B \implies A \leq_m^{p} B$.
2. $\leq_m^p$, $\leq_m^{\log}$, *and* $\leq_m^{\log\text{-lin}}$ *are reflexive and transitive.*
3. *If* $A \in \mathrm{P}$ *and* $B, \overline{B} \neq \emptyset$ *then* $A \leq_m^p B$.
4. *If* $A \in \mathrm{L}$ *and* $B, \overline{B} \neq \emptyset$ *then* $A \leq_m^{\log\text{-lin}} B$.

The proposition implies that non-trivial problems in P cannot be separated by means of $\leq_m^p$, i.e., $\leq_m^p$ is too coarse for P (and NL).

By experience, $\leq_m^p$-reductions between concrete problems can be replaced by $\leq_m^{\log}$. (Note that this is not a theorem!) That is, in the following we only consider $\leq_m^{\log}$ and $\leq_m^{\log\text{-lin}}$.

Closure of (complexity) class $\mathcal{K}$ under $\leq_m^r$:

- $\mathcal{R}_m^r(\mathcal{K}) =_{\text{def}} \{\ A \mid (\exists B \in \mathcal{K})[A \leq_m^r B]\ \}$
- $\mathcal{R}_m^r(B) =_{\text{def}} \mathcal{R}_m^r(\{B\}) = \{\ A \mid A \leq_m^r B\ \}$
- $\mathcal{K}$ is *closed under* $\leq_m^r \iff_{\text{def}} \mathcal{R}_m^r(\mathcal{K}) = \mathcal{K}$

Proposition 4.3 *Let $\mathcal{K}$ be any class of languages.*

1. *$\mathcal{R}_m^{\log}$ and $\mathcal{R}_m^{\log\text{-lin}}$ are hull operators (i.e., they are extensional, monotone, and idempotent).*
2. *$\mathcal{K} \subseteq \mathcal{R}_m^{\log\text{-lin}}(\mathcal{K}) \subseteq \mathcal{R}_m^{\log}(\mathcal{K})$.*
3. *If $\mathcal{K}$ is closed under $\leq_m^{\log}$ then $\mathcal{K}$ is closed under $\leq_m^{\log\text{-lin}}$.*

Theorem 4.4 *Let $X \in \{\mathrm{D}, \mathrm{N}\}$, $s(n) \geq \log n$ be space-constructible, and $t(n) \geq n$.*

1. $\mathcal{R}_m^{\log}(\mathrm{XSPACE}(s)) = \mathrm{XSPACE}(s(\mathrm{Pol}\ n))$.
2. $\mathcal{R}_m^{\log\text{-lin}}(\mathrm{XSPACE}(s)) = \mathrm{XSPACE}(s(O(n)))$.
3. $\mathcal{R}_m^{\log}(\mathrm{XTIME}(\mathrm{Pol}\ t)) = \mathrm{XTIME}(\mathrm{Pol}\ t(\mathrm{Pol}\ n))$.
4. $\mathcal{R}_m^{\log\text{-lin}}(\mathrm{XTIME}(\mathrm{Pol}\ t)) = \mathrm{XTIME}(\mathrm{Pol}\ t(O(n)))$.

Corollary 4.5

1. $\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}, \mathrm{PSPACE}, \mathrm{EXP}, \mathrm{NEXP}$ *are closed under* $\leq_m^{\log}$.
2. $\mathrm{LIN}, \mathrm{NLIN}, \mathrm{E}, \mathrm{NE}$ *are not closed under* $\leq_m^{\log}$.
3. $\mathrm{L}, \mathrm{NL}, \mathrm{P}, \mathrm{NP}, \mathrm{PSPACE}, \mathrm{EXP}, \mathrm{NEXP}, \mathrm{LIN}, \mathrm{NLIN}, \mathrm{E}, \mathrm{NE}$ *are closed under* $\leq_m^{\log\text{-lin}}$.

4.1.2 Complete problems

Definition 4.6 *Let $\mathcal{K}$ be closed under $\leq_m^r$ for $r \in \{\log, \log\text{-lin}\}$, and let B be any set.*

1. *B is* hard for $\mathcal{K}$ with respect to $\leq_m^r \iff_{\text{def}} \mathcal{K} \subseteq \mathcal{R}_m^r(B)$.
2. *B is* complete for $\mathcal{K}$ with respect to $\leq_m^r \iff_{\text{def}} \mathcal{K} = \mathcal{R}_m^r(B)$.

We also say that B is $\leq_m^r$-hard ($\leq_m^r$-complete) for $\mathcal{K}$.

Suppose a set B is $\leq_m^r$-complete for $\mathcal{K}$. Now let $C \in \mathcal{K}$ be another set such that $B \leq_m^r C$. Then, (by transitivity of $\leq_m^r$ and closedness of $\mathcal{K}$ under $\leq_m^r$) we obtain that C is $\leq_m^r$-complete for $\mathcal{K}$. This establishes a methodology for proving problems complete for a complexity class.

Proposition 4.7 *Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be closed under $\leq_m^r$, and let B be $\leq_m^r$-complete for $\mathcal{K}_1$. Then,*

$$\mathcal{K}_1 \subseteq \mathcal{K}_2 \Longleftrightarrow B \in \mathcal{K}_2.$$

Corollary 4.8 *Let B $\leq_m^r$-complete for $\mathcal{K}$.*

1. *If* $\mathcal{K} = \mathrm{NL}$ *then:* $\mathrm{L} = \mathrm{NL} \iff B \in \mathrm{L}$
2. *If* $\mathcal{K} = \mathrm{P}$ *then:* $\mathrm{NL} = \mathrm{P} \iff B \in \mathrm{NL}$
3. *If* $\mathcal{K} = \mathrm{NP}$ *then:* $\mathrm{P} = \mathrm{NP} \iff B \in \mathrm{P}$
4. *If* $\mathcal{K} = \mathrm{PSPACE}$ *then:* $\mathrm{NP} = \mathrm{PSPACE} \iff B \in \mathrm{NP}$
5. *If* $\mathcal{K} = \mathrm{coNP}$ *then:* $\mathrm{NP} = \mathrm{coNP} \iff B \in \mathrm{NP}$

Theorem 4.9

1. *There are $\leq_m^{\log}$-complete sets for* $\mathrm{NL}, \mathrm{P}, \mathrm{NP}, \mathrm{PSPACE}, \mathrm{EXP}, \mathrm{NEXP}$.
2. *There are $\leq_m^{\log\text{-lin}}$-complete sets for* $\mathrm{LIN}, \mathrm{NLIN}, \mathrm{E}, \mathrm{NE}$.
3. *There are no $\leq_m^{\log\text{-lin}}$-complete sets for* $\mathrm{PSPACE}, \mathrm{EXP}, \mathrm{NEXP}$.

Complete problems for the class NL:

Problem:	GAP (graph accessibility problem)
Input:	directed graph $G = (V, E)$, vertices $u, v \in V$
Question:	Is there a (u, v)-path in G?

Theorem 4.10 GAP *is $\leq_m^{\log}$-complete for* NL.

Corollary 4.11 $\mathrm{GAP} \in \mathrm{L} \Longleftrightarrow \mathrm{L} = \mathrm{NL}$.

Remark: UGAP (undirect graph accessbility problem) is in L by a theorem of Reingold from 2004.

Complete problems for the class P:

Problem:	CVP (circuit value problem)
Input:	logical circuit using $\{\wedge, \vee, \neg\}$-gates (of arbitrary fan-in), assignment z
Question:	Does the circuit evaluate to 1?

Theorem 4.12 CVP *is* $\leq_m^{\log}$*-complete for* P.

Complete problems for the class NP:

Lemma 4.13 *For each language $A \subseteq \Sigma^*$, it holds that $A \in$ NP if and only if there exist a set $B \in$ P and a polynomial p such that for all $x \in \Sigma^*$,*

$$x \in A \iff (\exists z)[\ |z| = p(|x|) \wedge (x, z) \in B\ ].$$

Problem:	Circuit Sat (circuit satisfiability)
Input:	logical circuit using $\{\wedge, \vee, \neg\}$-gates (without an assigment to the inputs)
Question:	Is there an assignment z to the inputs of C such that $C(z)$ evaluates to 1?

Theorem 4.14 Circuit Sat *is* $\leq_m^{\log}$*-complete for* NP.

Problem:	SAT (satisfiability)
Input:	propositional formula $H = H(x_1, \dots, x_n)$ over $\{\wedge, \vee, \neg\}$
Question:	Is there a truth assignment to $x_1, \dots, x_n$ making H true?

Problem:	3SAT
Input:	a CNF $H = H(x_1, \dots, x_n)$ with exactly 3 literals in each clause
Question:	Is there a truth assignment to $x_1, \dots, x_n$ making H true?

Theorem 4.15 SAT *and* 3SAT *are* $\leq_m^{\log}$*-complete for* NP.

Corollary 4.16 P $=$ NP $\iff$ Circuit Sat $\in$ P $\iff$ SAT $\in$ P $\iff$ 3SAT $\in$ P

What is the simplest NP-complete SAT version to reduce from?

Problem:	(k,ℓ)-SAT
Input:	a CNF $H = H(x_1,\dots,x_n)$ with exactly k literals in each clause such that each variable x_i occurs in exactly ℓ clauses as a literal
Question:	Is there a truth assignment to $x_1,\dots,x_n$ making H true?

Then, if $k \geq 3$ and $\ell \geq 4$ then (k,ℓ)-SAT is $\leq_m^{\log}$-complete for NP; otherwise it is P. So, a complexity jump occurs between $(3,3)$-SAT and $(3,4)$-SAT.

Corollary 4.17 SUBSET SUM *is* $\leq_m^{\log}$*-complete for* NP.

Problem:	TAUT (tautology)
Input:	propositional formula $H = H(x_1,\dots,x_n)$
Question:	Is H a tautology, i.e., is each truth assignment to $x_1,\dots,x_n$ a satisfying assignment for H?

Corollary 4.18 TAUT *is* $\leq_m^{\log}$*-complete for* coNP.

Complete problems beyond NP

Let Σ be an alphabet, $\|\Sigma\| \geq 2$. We define *regular expressions* over $\cup, \cdot, {}^*$:

- $\emptyset$ is an expression.
- If $a \in \Sigma$ then a is an expression.
- If H and H' are expressions then $H \cup H'$, $H \cdot H'$, and H^* are expressions.

A regular expression H defines a language $L(H)$ according to the following rules:

- $L(\emptyset) =_{\text{def}} \emptyset$.
- $L(a) =_{\text{def}} \{a\}$.
- $L(H \cup H') =_{\text{def}} L(H) \cup L(H')$.
- $L(H \cdot H') =_{\text{def}} \{\ xy \mid x \in L(H), y \in L(H')\ \} = L(H) \cdot L(H')$.
- $L(H^*) =_{\text{def}} L(H)^*$.

We consider the following inequivalence problem for regular expressions:

$$\mathrm{INEQ}(\Sigma, \cup, \cdot, ^*) =_{\mathrm{def}} \{\ (H, H') \mid L(H) \neq L(H')\ \}$$

We also discuss INEQ version for regular expressions defined by other operations, e.g., $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^2)$, $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^-)$.

Theorem 4.19 *1.* $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^*)$ *is* $\leq_m^{\log\text{-lin}}$*-complete for* NLIN.

2. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^*)$ *is* $\leq_m^{\log}$*-complete for* PSPACE.

3. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^2)$ *is* $\leq_m^{\log\text{-lin}}$*-complete for* NE.

4. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^2)$ *is* $\leq_m^{\log}$*-complete for* NEXP.

5. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^*, ^2)$ *is* $\leq_m^{\log\text{-lin}}$*-complete for* $\mathrm{DSPACE}(2^{O(n)})$.

6. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^*, \cap)$ *is* $\leq_m^{\log\text{-lin}}$*-complete for* $\mathrm{DSPACE}(2^{O(n)})$.

7. $\mathrm{INEQ}(\Sigma, \cup, \cdot, ^-)$ *is* $\leq_m^{\log}$*-hard for* $\mathrm{DSPACE}\left(2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}{\scriptstyle O(\log n)}\ )$

4.2 The counting method

Appropriate (but combinatorially hard) for concrete computational models.

As an example, we consider the set $S =_{\mathrm{def}} \{\ ww^R \mid w \in \{0,1\}^*\ \}$.

Theorem 4.20 $S \notin$ 1-T-DSPACE(s) *for* $s = o(n)$.

Theorem 4.21 $S \notin$ 2-T-DSPACE(s) *for* $s = o(\log n)$.

P versus NP 5

The possible outcomes of the $\mathrm{P} \overset{?}{=} \mathrm{NP}$ challenge are:

- $\mathrm{P} = \mathrm{NP}$ – find a polynomial algorithm for SAT!
- $\mathrm{P} \neq \mathrm{NP}$ – prove a superpolynomial lower bound for SAT!
- $\mathrm{P} \overset{?}{=} \mathrm{NP}$ is independent of (certain systems of) set theory

Most complexity theorists believe that $\mathrm{P} \neq \mathrm{NP}$.

Why is it hard to prove that $\mathrm{P} \neq \mathrm{NP}$?

- By *counting*: it is combinatorially involved, e.g., only $4n$ lower bound for SAT, and applicable only to concrete computational models that are typical for small complexity classes, e.g., circuit complexity.
- By *diagonalization*: it leads to relativizable results.

5.1 Oracle Turing machines

An *oracle Turing machine* M is a Turing machine equipped with an additional (one-way) oracle tape. M has special oracle states:

- $s_?$ is the query state, i.e., M asks a question z (the content of the oracle tape) to the oracle
- s_+ is the positive return state, i.e., oracles answer "yes" to the question (and the oracle tape is cleared)
- s_- is the negative return state, i.e., oracles answer "no" to the question (and the oracle tape is cleared)

The oracle answer appears in a single step!

The work of a deterministic oracle Turing machine on input x and for a general oracle B can be depicted by a *decision tree*:

If the oracle B is fixed then one path is realized.

For complexity considerations the work of the oracle is neglected, i.e., the space used on oracle tape for asking questions does not count for running space and asking an oracle question (and getting the answer) is counted as one step. (Note that an oracle is allowed to be undecidable.)

We define the *relativized complexity classes* (relative to an oracle B):

1. $\mathrm{DSPACE}^B(s), \mathrm{NSPACE}^B(s), \mathrm{DTIME}^B(\mathrm{Pol}\ t)\mathrm{NTIME}^B(\mathrm{Pol}\ t)$
2. $\mathrm{L}^B, \mathrm{NL}^B, \mathrm{P}^B, \mathrm{NP}^B, \mathrm{PSPACE}^B$.
3. Let $\mathcal{K}$ be a class of oracle sets. Then,

$$\begin{aligned} \mathrm{XSPACE}^{\mathcal{K}}(s) &=_{\mathrm{def}} \bigcup_{B\in\mathcal{K}} \mathrm{XSPACE}^B(s) \\ \mathrm{XTIME}^{\mathcal{K}}(t) &=_{\mathrm{def}} \bigcup_{B\in\mathcal{K}} \mathrm{XTIME}^B(t) \end{aligned}$$

All theorem relating complexity classes are true relative to an arbitrary oracle B:

- $\mathrm{DSPACE}^B(s) \subseteq \mathrm{NSPACE}^B(s) \subseteq \mathrm{DTIME}^B(2^{O(s)})$
- $\mathrm{DTIME}^B(\mathrm{Pol}\ t) \subseteq \mathrm{NTIME}^B(\mathrm{Pol}\ t) \subseteq \mathrm{DSPACE}^B(\mathrm{Pol}\ t)$
- $\mathrm{NSPACE}^B(s) \subseteq \mathrm{DSPACE}^B(s)$
- $\mathrm{coNSPACE}^B(s) = \mathrm{NSPACE}^B(s)$

Hierarchy theorems are, as well, true relative to an arbitrary oracle. We say that these theorems *relativize*. So, basically, diagonalization is a relativizable proof technique.

However, $\mathrm{P} \overset{?}{=} \mathrm{NP}$ cannot be solved using a relativizable proof technique.

5.2 P=NP relative to some oracle

Theorem 5.1 *There is an oracle B such that $\mathrm{P}^B = \mathrm{NP}^B$.*

5.3 P≠NP relative to some oracle

Theorem 5.2 *There is an oracle B such that $\mathrm{P}^B \neq \mathrm{NP}^B$.*

Remark: For the complexity classes IP and PSPACE, it is known that

- $\mathrm{IP} = \mathrm{PSPACE}$ and
- there is an oracle B such that $\mathrm{IP}^B \neq \mathrm{PSPACE}^B$.

The proof of $\mathrm{IP} = \mathrm{PSPACE}$ is based on the non-relativizable proof technique called *algebraization.*

6 The polynomial hierarchy

6.1 Turing reductions

Definition 6.1 *Let $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$ be sets. Then, A is said to be* polynomial-time Turing-reducible *to B (in symbols: $A \leq_T^p B$) if and only if $A \in \mathrm{P}^B$.*

Remark: $\leq_m^p$-reducibility is special case of $\leq_T^p$: one oracle query and no post-computation.

Proposition 6.2

1. *$A \leq_m^p B \Longrightarrow A \leq_T^p B$.*
2. *$\leq_T^p$ is reflexive and transitive.*
3. *$A \leq_T^p \overline{A}$.*

6.2 The oracle hierarchy

For $r : \mathbb{N} \to \mathbb{N}$ define the following classes:

$$\begin{aligned}
\mathrm{P}^B[r] &=_{\mathrm{def}} \text{class of all sets that can be accepted by a POTM asking } \leq_{a.e.} r(|x|) \\
&\qquad\ \ \text{queries to } B \text{ on input } x \\
\mathrm{P}^B[\mathrm{Pol}\ r] &=_{\mathrm{def}} \bigcup_{k\geq 1} \mathrm{P}^B[r^k] \\
\mathrm{P}^B[O(r)] &=_{\mathrm{def}} \bigcup_{k\geq 1} \mathrm{P}^B[k \cdot r]
\end{aligned}$$

We use obvious extensions to classes $\mathcal{K}$ of oracles.

Proposition 6.3 *For any $r : \mathbb{N} \to \mathbb{N}$ and any set B,*

$$\mathrm{P} = \mathrm{P}^B[0] \subseteq \mathrm{P}^B[r] \subseteq \mathrm{P}^B[\mathrm{Pol}\ n] = \mathrm{P}^B.$$

Definition 6.4 *The* polynomial hierarchy *consists of the following classes:*

1. $\Theta_0^p = \Delta_0^p = \Sigma_0^p = \Pi_0^p =_{\mathrm{def}} \mathrm{P}$.
2. $\Theta_{k+1}^p =_{\mathrm{def}} \mathrm{P}^{\Sigma_k^p}[O(\log n)]$, $\Delta_{k+1}^p =_{\mathrm{def}} \mathrm{P}^{\Sigma_k^p}$, $\Sigma_{k+1}^p =_{\mathrm{def}} \mathrm{NP}^{\Sigma_k^p}$, $\Pi_{k+1}^p =_{\mathrm{def}} \mathrm{co}\Sigma_{k+1}^p$.
3. $\mathrm{PH} =_{\mathrm{def}} \bigcup_{k\geq 0} \left(\Theta_k^p \cup \Delta_k^p \cup \Sigma_k^p \cup \Pi_k^p\right)$.

Theorem 6.5 *1.* $\Theta_1^p = \Delta_1^p = \mathrm{P}$, $\Sigma_1^p = \mathrm{NP}$, $\Pi_1^p = \mathrm{coNP}$.

2. $\mathrm{co}\Sigma_k^p = \Pi_k^p$, $\mathrm{co}\Delta_k^p = \Delta_k^p$, $\mathrm{co}\Theta_k^p = \Theta_k^p$.

3. $\Sigma_k^p \cup \Pi_k^p \subseteq \Theta_{k+1}^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

4. $\mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$.

The following inclusion diagram shows the structure of the polynomial hierarchy according to Theorem 6.5:

$$\begin{tikzcd}
& \mathrm{PSPACE} \arrow[d, dash] & \\
& \mathrm{PH} \arrow[d, dash, dotted] & \\
& \Sigma_2^p \cup \Pi_2^p \arrow[dl, dash] \arrow[dr, dash] & \\
\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}} \arrow[dr, dash] & & \Pi_2^p = \mathrm{coNP}^{\mathrm{NP}} \arrow[dl, dash] \\
& \Sigma_2^p \cap \Pi_2^p \arrow[d, dash] & \\
& \Delta_2^p = \mathrm{P}^{\mathrm{NP}} \arrow[d, dash] & \\
& \Theta_2^p = \mathrm{P}^{\mathrm{NP}}[O(\log n)] \arrow[d, dash] & \\
& \Sigma_1^p \cup \Pi_1^p = \mathrm{NP} \cup \mathrm{coNP} \arrow[dl, dash] \arrow[dr, dash] & \\
\Sigma_1^p = \mathrm{NP} \arrow[dr, dash] & & \Pi_1^p = \mathrm{coNP} \arrow[dl, dash] \\
& \mathrm{NP} \cap \mathrm{coNP} \arrow[d, dash] & \\
& \Theta_0^p = \Delta_0^p = \Sigma_0^p = \Pi_0^p = \Theta_1^p = \Delta_1^p = \mathrm{P} &
\end{tikzcd}$$

Proposition 6.6 *1.* Σ^p_k *is closed under* $\cap, \cup, \times, \leq^p_m$.

2. Π^p_k *is closed under* $\cap, \cup, \times, \leq^p_m$.

3. Θ^p_k *and* Δ^p_k *both are closed under* $\cap, \cup, \times, \leq^p_m$.

6.3 The quantifier hierarchy

For a set B, define the following sets:

$$\begin{aligned} B_\wedge &=_{\text{def}} \{ (x_1, \dots, x_m) \mid m \geq 1,\ x_1 \in B \wedge \dots \wedge x_m \in B \} \\ B_\vee &=_{\text{def}} \{ (x_1, \dots, x_m) \mid m \geq 1,\ x_1 \in B \vee \dots \vee x_m \in B \} \end{aligned}$$

Lemma 6.7 *For any set* B, $k \geq 0$, *the following holds:*

1. $B \in \Sigma^p_k \implies B_\wedge, B_\vee \in \Sigma^p_k$

2. $B \in \Pi^p_k \implies B_\wedge, B_\vee \in \Pi^p_k$

6.4 Complete problems

In the following, we use $\vec{x}_i$ to denote a sequence of propositional variables. We define quantified versions of the satisfiablity problem for a fixed number $k \in \mathbb{N}_+$ of alternating quantifier

$$\begin{aligned} \text{SAT}_k =_{\text{def}} \{ H \quad | \quad & H = H(\vec{x}_1, \dots, \vec{x}_k) \text{ is a propositional formula such that} \\ & (\exists z_1)(\forall z_2)(\exists z_3) \dots (Qz_k)[H(z_1, \dots, z_k)] \text{ is true } \} \end{aligned}$$

and the saitsfiability problem for *quantified boolean formulas*

$$\text{QBF} =_{\text{def}} \{ (H, k) \mid H \in \text{SAT}_k \}$$

Theorem 6.8 *1.* SAT_k *is* $\leq^{\log}_m$*-complete for* Σ^p_k *for* $k \in \mathbb{N}_+$.

2. $\overline{\text{SAT}_k}$ *is* $\leq^{\log}_m$*-complete for* Π^p_k *for* $k \in \mathbb{N}_+$.

3. QBF *is* $\leq^{\log}_m$*-complete for* PSPACE.

Alternation 7

Alternation is a further computational mode generalizing nondeterminism.

7.1 Alternating Turing machines

An *alternating Turing machine* is defined to be an NTM with the following types of states:

- an accepting halting state
- a rejecting halting state
- existential states
- universal states

The states of an ATM decompose into pairwise disjoint, exhausting sets with respect to the types. Accordingly, configurations consisting of tape inscriptions, head positions, and states are classified into accepting, rejecting, existential, or universal one's depending on the state.

Let $\beta_M(x)$ be the computation tree of an ATM M on input x, i.e., vertices represent configurations, children are successor configurations. Define an *accepting subtree* β of $\beta_M(x)$ to fulfill the following conditions:

- β contains the root of $\beta_M(x)$, i.e., the initial configuration,
- for each existential configuration in β, β contains exactly one child of $\beta_M(x)$,
- for each universal configuration in β, β contains all children of $\beta_M(x)$,
- leaves of β are accepting configurations.

Next we introduce the semantics for an ATM. We say that an ATM M

- accepts $x \Longleftrightarrow_{\text{def}}$ there exists an accepting subtree β of $\beta_M(x)$,
- accepts the set $L(M) =_{\text{def}} \{ x \mid M \text{ accepts } x \}$,
- accepts $L(M)$ in time $t : \mathbb{N} \to \mathbb{N} \Longleftrightarrow_{\text{def}}$ for each $x \in L(M)$, there exists an accepting subtree β of $\beta_M(x)$ of height $\leq t(|x|)$,
- accepts $L(M)$ in space $s : \mathbb{N} \to \mathbb{N} \Longleftrightarrow_{\text{def}}$ for each $x \in L(M)$, there exists an accepting subtree β of $\beta_M(x)$ such that all configurations in β are length-bounded by $s(|x|)$.

7.2 Alternating time

7.3 Alternating space

Probabilisitic complexity 8

8.1 Probabilistic Turing machines

The model of a *probabilistic Turing machine* is essentially the same like a nondeterministic Turing machine with the difference that only one computation paths is followed. In order to decide which path to follow, a fair coin is tossed for each nondeterministic branch. Thus, for a computation path r we obtain a certain probability

$$\mathrm{prob}(r) =_{\mathrm{def}} \left(\frac{1}{2}\right)^{\text{number of branches on } r}$$

8.2 The class BPP

8.3 The classes #P and GapP

8.4 The class PP

8.5 The class $\oplus$P

8.6 PH and PP

Epilogue: The graph isomorphism problem 9

Bibliography

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[BDG95] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I.* Texts in Theoretical Computer Science. 2nd edition. Springer-Verlag, Berlin, 1995.

[BDG90] José L. Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity II.* EATCS Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1990.

[BC93] Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall International Series in Computer Science. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, Cambridge, UK, 2009.

[HO01] Lane A. Hemaspaandra and Mitsunori Ogihara. *The Complexity Theory Companion.* An EATCS series. Springer-Verlag, Berlin, 2001.

[WW86] Klaus W. Wagner and Gerd Wechsung. *Computational Complexity.* Reidel, Dordrecht, 1986.

[Wec00] Gerd Wechsung. *Vorlesungen zur Komplexitätstheorie.* Teubner-Verlag, Stuttgart, 2000. In German.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation* 2nd edition. Addison Wesley, Reading, MA, 2001.