Complexity Theory
WS 2009/10

Prof. Dr. Erich Grädel

Mathematische Grundlagen der Informatik
RWTH Aachen

Contents

1 Deterministic Turing Machines and Complexity Classes 1
1.1 Turing machines 1
1.2 Time and space complexity classes 4
1.3 Speed-up and space compression 7
1.4 The Gap Theorem 9
1.5 The Hierarchy Theorems 11

2 Nondeterministic complexity classes 17
2.1 Nondeterministic Turing machines 17
2.2 Elementary properties of nondeterministic classes 19
2.3 The Theorem of Immerman and Szelepcsényi 21

3 Completeness 27
3.1 Reductions 27
3.2 NP-complete problems: SAT and variants 28
3.3 P-complete problems 34
3.4 NLOGSPACE-complete problems 38
3.5 A PSPACE-complete problem 42

4 Oracles and the polynomial hierarchy 47
4.1 Oracle Turing machines 47
4.2 The polynomial hierarchy 49
4.3 Relativisations 52

5 Alternating Complexity Classes 55
5.1 Complexity Classes 56
5.2 Alternating Versus Deterministic Complexity 57
5.3 Alternating Logarithmic Time 61

6 Complexity Theory for Probabilistic Algorithms 63
6.1 Examples of probabilistic algorithms 63
6.2 Probabilistic complexity classes and Turing machines 72
6.3 Probabilistic proof systems and Arthur-Merlin games 81

1 Deterministic Turing Machines and Complexity Classes

1.1 Turing machines

The simplest model of a Turing machine (TM) is the deterministic 1-tape Turing machine. Despite its simplicity, this model is sufficiently general to capture the notion of computability and allows us to define a very intuitive concept of computational complexity. During this course we will also use more general models of computation with the following facilities:

- a separate read-only input tape;
- a separate write-only output tape;
- more general types of memory, e.g., k linear tapes (for $k \geq 1$), higher-dimensional memory space, etc.

The corresponding definitions of configurations, computations, etc. need to be adjusted accordingly. We will do this for one specific model.

Definition 1.1. A *(deterministic) Turing machine with separate input and output tapes and k working tapes* is given by

$$M = (Q, \Gamma_{\text{in}}, \Gamma_{\text{out}}, \Sigma, q_0, F, \delta)$$

where

- Q is a finite set of states,
- Σ is the finite working alphabet, with a distinguished symbol $\square$ (blank),
- $\Gamma_{\text{in}}, \Gamma_{\text{out}}$ are the input and output alphabets (often $\Gamma_{\text{in}}, \Gamma_{\text{out}} = \Sigma$),
- $q_0 \subseteq Q$ is the initial state,
- $F \subseteq Q$ is the set of final states, and

- $\delta : (Q \setminus F) \times \Gamma_{\text{in}} \times \Sigma^k \to$
  $Q \times \{-1, 0, 1\} \times \Sigma^k \times \{-1, 0, 1\}^k \times (\Gamma_{\text{out}} \cup \{*\})$
  is the transition function.

A *configuration* is a complete description of all relevant data at a certain moment of the computation (state, memory contents, input, etc.). It is useful to distinguish between *partial* and *total* configurations.

Definition 1.2. Let M be a Turing machine. A *partial configuration* of M is a tuple $C = (q, w_1, \ldots, w_k, p_0, p_1, \ldots, p_k) \in Q \times (\Sigma^*)^k \times \mathbb{N}^{k+1}$, where

- q is the current state,
- $w_1, \ldots, w_k$ are the inscriptions on the working tapes,
- p_0 is the position on the input tape, and
- $p_1, \ldots, p_k$ are the positions on the working tapes.

The inscription of the ith working tape is given by a finite word $w_i = w_{i_0} \ldots w_{i_m} \in \Sigma^*$. There are only blanks on the fields $j > m$ of the infinite tape. When, in addition to a partial configuration, the inscriptions of the input and output tapes are given, one obtains a *total configuration* of M.

The *total initial configuration* $C_0(x)$ of M on $x \in \Gamma_{\text{in}}^*$ is given by

$$C_0(x) = (q_o, \varepsilon, \ldots, \varepsilon, 0, \ldots, 0, x, \varepsilon)$$

with

- the initial state q_0,
- empty working tapes, that is, $w_1 = w_2 = \cdots = w_k = \varepsilon$, (we denote the empty word by ε),
- position 0 on all tapes,
- the inscription x on the input tape, and
- the inscription ε on the output tape.

Remark 1.3. A *final configuration* is a configuration $C = (q, \overline{w}, \overline{p}, x, y)$ with $q \in F$. The word y (the inscription on the output tape) is the *output* of the final configuration C.

Successor configuration. Let $C = (q, w_1, \ldots, w_k, p_0, p_1, \ldots, p_k, x, y)$ be a (total) configuration of a Turing machine M. The transition

to the next configuration is determined by the value of the transition function δ on the current state q, and the values that have been read while in C, i.e., the symbols x_{p_0} read from the input tape and the symbols $w_{1\,p_1}, \ldots, w_{k\,p_k}$ read from the working tapes. Let $\delta(q, x_{p_0}, w_{1\,p_1}, \ldots, w_{k\,p_k}) = (q', m_0, a_1, \ldots, a_k, m_1, \ldots, m_k, b)$. Then $\Delta(C) := (q', \overline{w}', \overline{p}', x, y')$ is a successor configuration of C if

- w_i' results from w_i by replacing the symbol $w_{i\,p_i}$ with a_i,
- $p_i' = p_i + m_i$ (for $i = 0, \ldots, k$) and
- $y' = \begin{cases} y & \text{if } b = *, \\ yb & \text{if } b \in \Gamma_{\text{out}}. \end{cases}$

Notation. We write $C \vdash_M C'$, if $C' = \Delta(C)$.

Definition 1.4. A *computation* of M on x is a sequence $C_0, C_1, \ldots$ of (total) configurations of M with $C_0 = C_0(x)$ and $C_i \vdash_M C_{i+1}$ for all $i \geq 0$. The computation is *complete* if it is either infinite or it ends in a final configuration.

The *function computed by* M is a partial function $f_M : \Gamma_{\text{in}}^* \to \Gamma_{\text{out}}^*$. Thereby $f_M(x) = y$ iff the complete computation of M on x is finite and ends in a final configuration with output y.

Definition 1.5. A *k-tape acceptor* is a k-tape Turing machine M ($k \geq 1$), whose final states F are partioned into a set F^+ of *accepting* states and a set F^- of *rejecting* states. *M accepts x*, iff the computation of M on x halts in a state $q \in F^+$. *M rejects x*, iff the computation of M on x halts in a state $q \in F^-$.

Definition 1.6. Let $L \subseteq \Gamma_{\text{in}}^*$ be a language. *M decides L* if M accepts all $x \in L$ and rejects all $x \in \Gamma_{\text{in}}^* \setminus L$. L is *decidable* if there exists an acceptor M that decides L. We will write $L(M)$ to denote the set of inputs accepted by M.

In the following, we will often also use k-tape Turing machines without distinguished input and output tapes. In these cases the first working tape will also be the input tape while some other tape (or tapes) will overtake the role of the output tape.

Conventions. As long as not specified in a different way:

- a Turing machine (TM) shall be a k-tape Turing machine (for every $k \geq 1$), where k denotes the total number of tapes, possibly including separate input and output tapes;
- Γ shall stand for the input alphabet.

1.2 Time and space complexity classes

Definition 1.7. Let M be a Turing machine and x some input. Then $\text{time}_M(x)$ is the length of the complete computation of M on x and $\text{space}_M(x)$ is the total number of working tape cells used in the computation of M on x. Let $T, S : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be monotonically increasing functions. A TM M is

- *T-time bounded* if $\text{time}_M(x) \leq T(|x|)$ for all inputs $x \in \Gamma^*$, and
- *S-space bounded* if $\text{space}_M(x) < S(|n|)$ for all inputs $x \in \Gamma^*$.

Definition 1.8.

(i) $\textsc{Dtime}_k(T)$ is the set of all languages L for which there exists a T-time bounded k-tape TM that decides L.
(ii) $\textsc{Dspace}_k(S)$ is the set of all languages L for which there exists a S-space bounded k-tape TM that decides L.
(iii) $\textsc{Dtime}(T) = \bigcup_{k \in \mathbb{N}} \textsc{Dtime}_k(T)$.
(iv) $\textsc{Dspace}(S) = \bigcup_{k \in \mathbb{N}} \textsc{Dspace}_k(S)$.
(v) $\textsc{Dtime-space}_k(T, S)$ is the set of all languages L for which there is a T-time bounded and S-space bounded k-tape TM that decides L.
(vi) $\textsc{Dtime-space}(T, S) = \bigcup_{k \in \mathbb{N}} \textsc{Dtime-space}_k(T, S)$.

Important complexity classes are:

- $\textsc{Logspace} := \bigcup_{d \in \mathbb{N}} \textsc{Dspace}(d \log n)$,
- $(\textsc{Ptime} =)\ \text{P} := \bigcup_{d \in \mathbb{N}} \textsc{Dtime}(n^d)$,
- $\textsc{Pspace} := \bigcup_{d \in \mathbb{N}} \textsc{Dspace}(n^d)$,
- $\textsc{Exptime} := \bigcup_{d \in \mathbb{N}} \textsc{Dtime}(2^{n^d})$,
- $\textsc{Expspace} := \bigcup_{d \in \mathbb{N}} \textsc{Dspace}(2^{n^d})$.

Attention: Some authors may also define $\textsc{Exptime}$ as $\bigcup_{d \in \mathbb{N}} \textsc{Dtime}(2^{dn})$.

Elementary observations on the relationship between time and space complexity lead to the following statements.

Theorem 1.9.

(a) $\textsc{Dtime}(T) \subseteq \textsc{Dspace}(O(T))$ for all functions $T : \mathbb{N} \to \mathbb{N}$.

(b) $\textsc{Dspace}(S) \subseteq \textsc{Dtime}\big(2^{O(S)}\big)$ for all functions $S\colon \mathbb{N} \to \mathbb{N}$ with $S(n) \geq \log n$.

Proof. (a) A k-tape Turing machine can visit at most k fields in one step. (b) Because $L \in \textsc{Dspace}(S)$, we can assume that L is decided by a TM M with one input tape and k working tapes using space S.

For every input x (and $n = |x|$), any partial configuration is obtained at most once during the computation of M on x. Otherwise, M would run in an endless loop and could not decide L. The number of partial configurations with space $S(n)$ is bounded by

$$|Q| \cdot (n+1) \cdot S(n)^k \cdot |\Sigma|^{S(n)} = 2^{O(S(n))}, \text{ whenever } S(n) \geq \log n.$$

Here, $(n+1)$ is the number of possible positions of the input tape, $S(n)^k$ the number of positions of the working tapes and $|\Sigma|^{k \cdot S(n)}$ the number of possible memory contents. Thus, $\text{time}_M(x) \leq 2^{O(S(n))}$. Q.E.D.

Corollary 1.10. $\textsc{Logspace} \subseteq \textsc{P} \subseteq \textsc{Pspace} \subseteq \textsc{Exptime}$.

Theorem 1.11 (Tape reduction)**.** Let $S(n) \geq n$. Then

$$\textsc{Dtime-space}(T, S) \subseteq \textsc{Dtime-space}_1(O(T \cdot S), S).$$

Proof. (Simulation of a k-tape TM using a 1-tape TM.) Let M be a T-time bounded and S-space bounded k-tape TM that decides L. The idea is to simulate the k tapes of M using $2k$ tracks on a single tape of a 1-tape TM M'. Track $2j-1$ of the tape of M' will contain the inscription on tape j of M and track $2j$ a mark $(*)$ at the current head position of tape j of M.

Before simulating a single step of M, the head of M' is at the first (leftmost) mark. The simulation is accomplished in three phases.

(i) M' moves to the right up to the last mark and saves (in the state set) the symbols at the current positions of M, that is, the information needed to determine the transition of M. Time needed: at most $S(n)$ steps.

(ii) M' determines the transition taken by M. This takes one step.

(iii) M' returns to the first mark performing on its way back all necessary changes on the tape. Time needed: at most $S(n)$ steps.

M' accepts (or rejects) iff M accepts (or rejects). At most $S(n)$ fields contain information. Therefore, the marks are at most $S(n)$ fields apart. The simulating 1-tape TM thus needs $O(S(n))$ steps and no additional memory to simulate a step of M. The claim follows. Q.E.D.

Where did we use that $S(n) \geq n$? Consider an S-space bounded 2-tape Turing machine M, where $S(n) < n$ and where the first tape is a separate input tape. As long as M is reading the whole input, the simulating 1-tape TM will have to go to the rightmost position on the first tape to set the marks. This way, the two marks can be more than $S(n)$ fields away from each other.

Corollary 1.12. $\textsc{Dtime}(T) \subseteq \textsc{Dtime}_1(O(T^2))$.

This follows from Theorem 1.11 using the fact that $\text{space}_M(x) \leq O(\text{time}_M(x))$ for all M and all x. We also obtain:

Corollary 1.13. $\textsc{Dspace}(S) \subseteq \textsc{Dspace}_1(S)$ for $S(n) \geq n$.

To simulate a k-tape TM using a 2-tape TM, the time complexity increases by a logarithmic factor only.

Theorem 1.14. $\textsc{Dtime}(T) \subseteq \textsc{Dtime}_2(O(T \cdot \log T))$ for $T(n) \geq n$.

Proof (Outline). A k-tape TM M is simulated using a 2-tape TM M':

- 2 tracks on the first tape of M' are created for every tape of M.
- The second tape of M' is only used as intermediate memory for copy operations.

The first tape of M' is divided into blocks $\ldots, B_{-i}, B_{-i+1}, \ldots, B_{-1}, B_0, B_1, \ldots, B_i$, where $|B_0| = 1, |B_j| = 2^{|j|-1}$ for $j \neq 0$. All characters currently read by M can be found in block B_0. If the head on one track of M moves to the left, M' moves the entire inscription on the corresponding tapes to the right. This way, the current character will again be at position B_0. A clever implementation of this idea leads to a simulation with at most logarithmic time loss: if M is T-time bounded, then M' is $O(T \cdot \log T)$-time bounded. Q.E.D.

The complete proof can be found, e.g., in J. Hopcraft, J. Ullmann: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley 1979, pp. 292–295.

1.3 Speed-up and space compression

Definition 1.15. For functions $f, g : \mathbb{N} \to \mathbb{R}$, we write $f = o(g)$ to denote $\lim_{n\to\infty} f(n)/g(n) = 0$.

Theorem 1.16 (Speed-up theorem)**.**

$$\textsc{Dtime}_k(T) \subseteq \textsc{Dtime}_k(\max(n, \varepsilon \cdot T(n)))$$

for all $k > 1$, $\varepsilon > 0$, and $T\colon \mathbb{N} \to \mathbb{R}^{\geq 0}$ with $n = o(T(n))$.

Proof. Let M be a k-tape TM that decides L in time $T(n)$. Choose m in such way that $\varepsilon \cdot m \geq 16$. Let Σ be the working alphabet of M. We will construct a k-tape TM M' that uses the working alphabet $\Sigma \cup \Sigma^m$ so that it can encode m symbols of M by a single symbol. This way the computation can be speeded up.

(1) M' copies the input to a different tape compressing m symbols into one. Then, M' treats this working tape as the input tape. Time needed: n steps for copying and $\lceil \frac{n}{m} \rceil$ steps to return the head to the first symbol of the compressed input.

(2) M' simulates m steps of M taking 8 steps at a time. The following operations are executed on the working tapes:

(a) M' saves the contents of both neighboring fields of the current field "in the state set". This needs 4 steps: one to the left, two to the right, and one to the left again.

(b) M' determines the result of the next m steps of M. This is hard-coded in the transition function of M'. In m steps, M can only use or change fields that are at most m steps away from each other. In other words, it can only visit the current field of M' and both neighboring fields. Hence, M' needs 4 steps to implement this change.

(c) M' accepts or rejects iff M accepts or rejects, respectively.

Let x be an input of length n. Then

$$\text{time}_{M'}(x) \leq n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil \leq n + n/m + 8T(n)/m + 2.$$

Since $n = o(T(n))$, for every $d > 0$, there is an n_d so that $T(n)/n \geq d$ for all $n \geq n_d$. Therefore, $n \leq T(n)/d$ for $n \geq n_d$. For $n \geq max(2, n_d)$, we obtain $2n \geq n + 2$. Thus, M' needs at most

$$2n + \frac{n}{m} + 8\frac{T(n)}{m} \leq T(n)\left(\frac{2}{d} + \frac{1}{md} + \frac{8}{m}\right) = T(n)\left(\frac{2m + 8d + 1}{md}\right)$$

steps. Set $d = \frac{2m+1}{8}$. Then the number of steps of M' is bounded by

$$T(n)\left(\frac{8(2m + 1 + 2m + 1)}{m(2m + 1)}\right) = \frac{16}{m}T(n) \leq \varepsilon T(n)$$

for all $n \geq max(2, n_d)$. The finite number of inputs of length $< n_d$ can be accepted in n_d time. Q.E.D.

Corollary 1.17.

$$\textsc{Dtime}(T(n)) = \textsc{Dtime}(max(n, \varepsilon \cdot T(n))$$

for all $T : \mathbb{N} \to \mathbb{R}$ with $n = o(T(n))$ and all $\varepsilon > 0$.

A similar but easier proof shows the following.

Theorem 1.18 (Space compression)**.**

$$\textsc{Dspace}(S) \subseteq \textsc{Dspace}(max(1, \varepsilon \cdot S(n))$$

for all functions $S : \mathbb{N} \to \mathbb{R}^{\geq 0}$ and all $\varepsilon > 0$.

1.4 The Gap Theorem

In this and the following section, we address the question whether one is able to solve more problems when more ressources are provided. If S_2 increases faster than S_1, does this mean that $\textsc{Dspace}(S_2) \supsetneq \textsc{Dspace}(S_1)$ (and analogously for time)? We will show that this does not hold in the general case. Towards this, we will first prove the following lemma.

Lemma 1.19. Let M be a k-tape acceptor with $\max_{|x|=n} \text{space}_M(x) \leq S(n)$ for almost all n (that is, all but finitely many) and let $L(M)$ be the set of all inputs accepted by M. Then, $L(M) \in \textsc{Dspace}(S)$.

Proof. We build a k-tape acceptor M' such that $L(M') = L(M)$ and $space_{M'}(x) \leq S(|x|)$ for all x. The set $X = \{x \ : \ \text{space}_M(x) > S(|x|)\}$ is finite by definition. Hence, for inputs $x \in X$, we can hard-code the answer to $x \in L(M)$ in the transition function M' without using additional space. Q.E.D.

Theorem 1.20 (Gap Theorem)**.** For any computable total function $g : \mathbb{N} \to \mathbb{N}$ with $g(n) \geq n$, there exists a computable function $S : \mathbb{N} \to \mathbb{N}$ such that $\textsc{Dspace}(S) = \textsc{Dspace}(g \circ S)$.

Proof. Let $M_0, M_1, \ldots$ be a recursive enumeration of all Turing machines. Consider the function $S_i(n) := \max_{|x|=n} \text{space}_{M_i}(x) \cup \{\infty\}$ which returns the space required by Turing machine M_i on words of length n.

Lemma 1.21. The set $R := \{(i, n, m) \ : \ S_i(n) = m\}$ is decidable.

Proof. For every triple (i, n, m), there is a time bound $t \in \mathbb{N}$ on computations of M_i which, on inputs of length at most n, use at most m tape

Algorithm 1.1. $S(n)$

Input: n
$y := 1$
while *there is an* $i \leq n$ *with* $(i, n, y) \in P$ **do**
  choose the smallest such i
  $y := S_i(n)$
endwhile
$S(n) := y$

cells while no configuration occurs more than once. This bound t is computable from (i, n, m). By simulating t steps of M_i on the (finitely many) different inputs of length n, one can decide whether $S_i(n) = m$. Q.E.D.

We will use this result to construct a function $S : \mathbb{N} \to \mathbb{N}$ such that, for every $i \in \mathbb{N}$, either $S_i(n) \leq S(n)$ for almost all n, or $S_i(n) \geq g(S(n))$ for infinitely many n. Towards this, consider the set $P := \{(i, n, y) \in \mathbb{N}^3 \ : \ y < S_i(n) \leq g(y)\}$. By Lemma 1.21 and since g is computable, we obtain that P is decidable. Let $S : \mathbb{N} \to \mathbb{N}$ be the function defined by Algorithm 1.1. As P is decidable, S is a computable total function. It remains to show that

$$\textsc{Dspace}(g \circ S) \setminus \textsc{Dspace}(S) = \varnothing.$$

For any $L \in \textsc{Dspace}(g \circ S)$ we have $L = L(M_i)$ for some $i \in \mathbb{N}$. As $L \in \textsc{Dspace}(g \circ S)$, by definition $S_i(n) \leq g(S(n))$ holds for all $n \in \mathbb{N}$. The way S was constructed, we have $S_i(n) \leq S(n)$ for all $n \geq i$. Otherwise $S(n) < S_i(n) \leq g(S(n))$ would hold for some $i \leq n$, which is excluded by the algorithm. Hence, $S_i(n) \leq S(n)$ for almost all n and, according to Lemma 1.19, we can conclude that $L = L(M_i) \in \textsc{Dspace}(S)$. Q.E.D.

Application. Consider $g(n) = 2^n$. There exists a function S such that $\textsc{Dspace}(2^S) = \textsc{Dspace}(S)$. That is, using more space does not necessarily allow to solve more problems.

Analogously, one can show the follwing theorem on time complexity.

Theorem 1.22 (Gap Theorem for time complexity)**.** For every computable function g, there exists a computable function T with $\textsc{Dtime}(T) = \textsc{Dtime}(g \circ T)$.

Hence, there are computable functions f, g, h so that,

- $\textsc{Dtime}(f) = \textsc{Dtime}(2^f)$.
- $\textsc{Dtime}(g) = \textsc{Dtime}(2^{2^g})$.
- $\textsc{Dtime}(h) = \textsc{Dtime}\left(2^{\overset{\left.\right\}h(n) \text{ times}}{\cdot^{\cdot^{\cdot^{2}}}}}\right)$.

1.5 The Hierarchy Theorems

In the previous section, we have shown that increasing complexity bounds does not always allow us to solve more problems. We will now investigate under which conditions a complexity class is fully contained in another one. As in the proof of the undecidability of the Halting Problem for Turing machines, we will use a diagonalization argument. The proof will be kept very general with a view to complexity measures beyond time and space.

Let $\mathcal{M}$ be a class of abstract machines (e.g., 2-tape Turing machines) and R a ressource defined for machines in $\mathcal{M}$ (e.g., time or space) such that, for every machine $M \in \mathcal{M}$ and every input x, $R_M(x) \in \mathbb{N} \cup \{\infty\}$ is defined. For a function $T\colon \mathbb{N} \to \mathbb{N}$, $R(T)$ denotes the complexity class of all problems that machines in $\mathcal{M}$ with T-bounded ressource R can decide:

$$R(T) = \{L : \text{there is an } M \in \mathcal{M} \text{ deciding } L \text{ with } R_M(x) \leq T(|x|) \text{ for all } x\}.$$

Furthermore, we assume that there is an function ρ encoding machines in $\mathcal{M}$ over the alphabet $\{0,1\}$ in such way that the structure and computational behavior of M can be extracted effectively from $\rho(M)$.

Let $T, t\colon \mathbb{N} \to \mathbb{N}$ be functions, $\mathcal{M}_1$ and $\mathcal{M}_2$ classes of acceptors,

and R, r ressources defined for $\mathcal{M}_1$ and $\mathcal{M}_2$. We thus obtain the complexity classes $R(T)$ and $r(t)$.

Definition 1.23. $R(T)$ allows *diagonalization* over $r(t)$ if there exists a machine $D \in \mathcal{M}_1$ such that:

- D is T-bounded in ressource R and stops on every input. In other words, $L(D) \in R(T)$.
- For every machine $M \in \mathcal{M}_2$ that is t-bounded in ressource r,

  $$\rho(M)\#x \in L(D) \Leftrightarrow \rho(M)\#x \notin L(M).$$

  holds for almost all $x \in \{0,1\}^*$.

Theorem 1.24 (General Hierarchy Theorem)**.** If $R(T)$ allows diagonalization over $r(t)$, then $R(T) \setminus r(t) \neq \varnothing$.

Proof. Let D be the diagonalization machine from Definition 1.23. We will show that $L(D) \notin r(t)$. Otherwise, there would be a machine M that is t-bounded in ressource r with $L(D) = L(M)$. This, however, is impossible since for almost all x:

$$\rho(M)\#x \in L(D) \Leftrightarrow \rho(M)\#x \notin L(M)$$

holds. Therefore, $L(M) \neq L(D)$. Q.E.D.

Definition 1.25. A function $T : \mathbb{N} \to \mathbb{N}$ is called fully *time constructible* if there exists a Turing machine M such that $\text{time}_M(x) = T(|x|)$ for all x. Similarly, $S : \mathbb{N} \to \mathbb{N}$ is fully *space constructible* if $\text{space}_M(x) = S(|x|)$ holds for some Turing machine M and all x.

Time and space constructible functions are "proper" functions whose complexity is not much larger than their values. Most of the functions we usually consider are fully time and space constructible. Specifically, this is true for n^k, 2^n and $n!$. If two functions f and g have this property, the functions $f + g$, $f \cdot g$, 2^f and f^g do as well.

Theorem 1.26. Let $T, t : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that $T(n) \geq n$, with T time constructible and $t = o(T)$. Then $\text{DTIME}_k(t) \subsetneq \text{DTIME}(T)$ for all $k \in \mathbb{N}$.

Proof. We will show that $\textsc{Dtime}(T)$ allows diagonalization over $\textsc{Dtime}_k(t)$. Towards this, let D be a TM with the following properties:

(a) If M is a k-tape TM and $x \in \{0,1\}^*$, then, on input $\rho(M)\#x$, the machine D simulates the computation of M on $\rho(M)\#\rho(x)$.

(b) For each M, there is a constant c_M such that D needs at most c_M steps for the simulation of each step of M.

(c) At the same time, D simulates another TM N on separate tapes which executes precisely $T(n)$ steps on inputs of length n. By time constructability of T, such a machine exists.

(d) After $T(n)$ ($n = |\rho(M)\#x|$) steps, D stops and accepts $\rho(M)\#x$ iff the simulated computation of M on $\rho(M)\#x$ has rejected. Otherwise, if M has already accepted or the computation has yet not been completed, D rejects.

Let $L(D) = \{\rho(M)\#x \ : \ D \text{ accepts } \rho(M)\#x\}$. We have:

- $L(D) \in \textsc{Dtime}(T)$.
- For all M: $T(n) \geq c_M \cdot t(n)$ for almost all n (since $t = o(T)$). Therefore, D can simulate the computation of M on $\rho(M)\#x$ for almost all inputs $\rho(M)\#x$ in $T(n)$ steps.

Thus, $\rho(M)\#x \in L(D) \iff \rho(M)\#x \notin L(M)$. The claim follows from the General Hierarchy Theorem. Q.E.D.

Corollary 1.27 (Time Hierarchy Theorem)**.** Let $T(n) \geq n$, T be time-constructible and $t \cdot \log t = o(T)$. Then $\textsc{Dtime}(t) \subsetneq \textsc{Dtime}(T)$.

Proof. By Theorem 1.14, there is a constant c such that $\textsc{Dtime}(t) \subseteq \textsc{Dtime}_2(c \cdot t \cdot \log t)$. If $t \cdot \log t = o(T)$, then also $c \cdot t \cdot \log t = o(T)$ holds. Thus, by Theorem 1.26, there is a language

$$L \in \textsc{Dtime}(T) \setminus \textsc{Dtime}_2(c \cdot t \cdot \log t) \subseteq \textsc{Dtime}(T) \setminus \textsc{Dtime}(t). \text{ Q.E.D.}$$

Applications. As $T(n) = n^{d+1}$ is time-constructible for each $d \in \mathbb{N}$ and $t(n) = n^d \log n^d = O(n^d \log n) = o(n^{d+1}) = o(T(n))$, the following

holds:

$$\textsc{Dtime}(n^d) \subsetneq \textsc{Dtime}(n^{d+1}).$$

Corollary 1.28. For any time constructible and increasing function f with $\lim_{n\to\infty} f(n) = \infty$, the class P of all problems decidable in polynomial time is strictly included in $\textsc{Dtime}(n^{f(n)})$. In particular, $\mathrm{P} \subsetneq \textsc{Exptime}$.

Theorem 1.29 (Space Hierarchy Theorem)**.** Let $S, s : \mathbb{N} \to \mathbb{N}$ be two functions where S is space constructible and $S(n) \geq \log n$ and $s = o(S)$. Then, $\textsc{Dspace}(S) \setminus \textsc{Dspace}(s) \neq \varnothing$.

Proof. As we can reduce, by Theorem 1.11, the number of working tapes to one without increasing the space complexity, it is sufficient to consider a TM with one input and one working tape. Since M is s-space bounded, there are at most $|Q| \cdot (n+1) \cdot |\Sigma|^{s(n)} \cdot s(n) = (n+1)2^{O(s(n))}$ different partial configurations of M. The machine M therefore stops either after $\leq t(n) = 2^{c_M s(n) + \log(n+1)}$ steps or it never halts. Here, c_M denotes a constant which depends on M but not on n. Since S is space-constructible, there is a TM N with $\mathrm{space}_N(x) = S(|x|)$ for all x. It remains to show that $\textsc{Dspace}(S)$ allows diagonalization over $\textsc{Dspace}(s)$. Consider the machine D that operates on input $\rho(M)\#x$ as follows:

(a) At first, mark a range of $S(n)$ fields, by simulation of N. All subsequent operations will take place within this range. If other fields are accessed during the execution, D immediately stops and rejects the input.
(b) D initializes a counter to $t(n)$ and stores it on an extra tape.
(c) D simulates the computation of M on $\rho(M)\#x$ and decrements the counter at every simulated step.
(d) If the simulation accesses a non-marked field or M does not stop within $t(n)$ steps, then D rejects the input $\rho(M)\#x$. D also rejects if M accepts the input $\rho(M)\#x$. If D completes the simulation of a rejecting computation of M on $\rho(M)\#x$, then D accepts $\rho(M)\#x$.

We obtain:

- $L(D) \in \text{DSPACE}(S)$.
- It remains to show: if M is s-space bounded, then D can simulate the computation of M on $\rho(M)\#x$ for almost all x completely (or $t(n)$ steps of it).
  - Because $t(n) = 2^{O(s(n)+\log n)}$, $s = o(S)$ and $S(n) \geq \log n$, the counter $t(n)$ can be encoded by a word of length $S(n)$ for all n that are large enough.
  - Assuming M has an alphabet with d different symbols. Then D needs $\leq \log d$ fields to encode a symbol of M (note that this factor only depends on M but not on the input length).
  - For the simulation of M, the machine D needs at most space $\log d \cdot \text{space}_M(\rho(M)\#x) \leq \log d \cdot s(n) \leq S(n)$ for almost all n.

  For all sufficiently large x, the following holds: $\rho(M)\#x \in L(D) \iff \rho(M)\#x \notin L(M)$. Therefore, $\text{DSPACE}(S)$ allows diagonalization over $\text{DSPACE}(s)$. The claim follows with the General Hierarchy Theorem. Q.E.D.

Remark 1.30. As an immediate consequence we obtain $\text{LOGSPACE} \subsetneq \text{PSPACE}$. Thus, at least one of the inclusions $\text{LOGSPACE} \subseteq \text{P} \subseteq \text{PSPACE}$ must be strict. However, at the present time, we do not know whether $\text{LOGSPACE} \subsetneq \text{P}$ or $\text{P} \subsetneq \text{PSPACE}$.

2 Nondeterministic complexity classes

2.1 Nondeterministic Turing machines

Nondeterministic Turing machines (NTM) are defined just as their deterministic counterparts except that the transition function generally allows several possible transitions.

Again, the most important model is the k-tape TM. The possible transitions are given by a function $\delta : Q \times \Sigma^k \to \mathcal{P}(Q \times \Sigma^k \times \{-1, 0, 1\}^k)$ (modified accordingly if the first tape is a read-only input tape). Again, we will mainly deal with *acceptors*, so that the set of final states is partitioned $F = F^+ \cup F^-$.

A configuration of a nondeterministic Turing machine M usually has several successor configurations. Let $\mathrm{Next}(C) = \{C' : C \vdash_M C'\}$ be the set of successor configurations of C. For each NTM M there is an integer $r \in \mathbb{N}$ such that $|\mathrm{Next}(C)| \leq r$ for all configurations C of M. Given an input x for a nondeterministic Turing machine M, instead of a (complete) sequential computation, we consider a *computation tree* $\mathfrak{T}_{M,x}$ defined as follows:

- the root of $\mathfrak{T}_{M,x}$ is the initial configuration $C_0(x)$;
- the children of each node C are the elements of $\mathrm{Next}(C)$.

A computation path of M on x or, simply, a computation is a sequence $C_0, \ldots, C_t$ of configurations with $C_0 = C_0(x)$ and $C_{i+1} \in \mathrm{Next}(C_i)$, that is, a path through $\mathfrak{T}_{M,x}$ starting at the root.

Definition 2.1. A nondeterministic Turing machine is T-time bounded (respectively S-space bounded) if *no* computation M on inputs of length n takes more than $T(n)$ steps (uses more than $S(n)$ fields).

Definition 2.2. Let M be a NTM and x its input. *M accepts x*, if there is at least one computation of M on x that stops in an accepting configuration. $L(M) = \{x : M \text{ accepts x}\}$ is the language accepted by M.

Definition 2.3.
$\text{NTIME}(T) := \{L : \text{there is a } T\text{-time bounded NTM with } L(M) = L\}$.
$\text{NSPACE}(S) := \{L : \text{there is an } S\text{-space bounded NTM with } L(M) = L\}$.
Other classes such as $\text{NTIME}_k(T)$ can be defined analogously.

Remark 2.4. In informal descriptions of nondeterministic Turing machines, nondeterministic steps are often called "guesses". Thus, "Guess a $y \in \Sigma^m$ " means: Perform a sequence of m nondeterministic steps so that in the ith step, the ith symbol of y is nondeterministically chosen from $|\Sigma|$. The (pseudo-)instruction is equivalent to a computation tree of depth m with $|\Sigma|$ many successors at all inner nodes and with $|\Sigma|^m$ leaves labelled with $y \in \Sigma^m$.

Example 2.5 (A nondeterministic algorithm for the Reachability problem). The following algorithm solves REACHABILITY nondeterministically:

Algorithm 2.1 Nondeterministic REACHABILITY

Input: $G = (V, E)$, a directed graph, $a, b \in V$ $(|V| = n)$
$x := a$
for *n steps* **do**
  if $x = b$ **then** accept **else**
    guess $y \in V$ with $(x, y) \in E$
    $x := y$
  endif
endfor
reject

If there is a path in G from a to b, then there is also one of length $\leq n$ (longer paths would include cycles). Therefore, the algorithm has an accepting computation iff there is a path from a to b. The required space is $\leq 3 \cdot \log n$ ($\log n$ for x, y and the counter). Hence, REACHABILITY belongs to the complexity class $\text{NLOGSPACE} = \text{NSPACE}(O(\log n))$. As we have seen in the exercises, REACHABILITY also belongs to $\text{DSPACE}(O(\log^2 n))$.

2.2 Elementary properties of nondeterministic classes

In order to compare deterministic and nondeterministic complexity classes, we often look at the configuration graphs of nondeterministic Turing machines.

Definition 2.6. Let M be a nondeterministic Turing machine and $s \in \mathbb{N} \cup \{\infty\}$. Then

$$\mathrm{Conf}[s] := \{C : C \text{ is a configuration of } M \text{ using space at most } s\},$$

and the successor relation on configuration defines the (directed) *configuration graph* $G[M, s] = (\mathrm{Conf}[s], \vdash_M)$.

For any S space bounded nondeterministic TM M and any input x (with $|x| = n$), we have:

- The computation tree $\mathfrak{T}_{M,x}$ corresponds to the unravelling of $G[M, S(n)]$ from the input configuration $C_0(x)$.
- M accepts x if there is a path from $C_0(x)$ to some accepting configuration C_a in $G[M, S(n)]$.

Theorem 2.7. $\textsc{Dtime}(T) \subseteq \textsc{Ntime}(T) \subseteq \textsc{Nspace}(T) \subseteq \textsc{Dtime}(2^{O(T)})$ for all space-constructible $T : \mathbb{N} \to \mathbb{R}^+$ with $T(n) \geq \log n$.

Proof. The first inclusions $\textsc{Dtime}(T) \subseteq \textsc{Ntime}(T) \subseteq \textsc{Nspace}(T)$ are trivial. To prove the remaining inclusion, let M be a nondeterministic, T-space bounded TM. Since every configuration uses at most $T(n)$ fields, $G[M, T(n)]$ consists of at most $2^{T(n)}$ different configurations. M accepts x iff there is a path in G from $C_0(x)$ to an accepting configuration. In time $2^{O(T(n))}$, a deterministic algorithm can

(a) construct G and
(b) decide for all accepting configurations C_a whether G contains a path from $C_0(x)$ to C_a.

This follows from the fact that Reachability can be solved by a deterministic algorithm in polynomial time. Q.E.D.

Theorem 2.8 (Savitch's Theorem)**.** $\textsc{Nspace}(S) \subseteq \textsc{Dspace}(S^2)$ for any space constructible function $S(n) \geq \log n$.

Algorithm 2.2. **Reach**(C_1, C_2, k)

```
if k = 0 then
    if C_1 = C_2 ∨ C_2 ∈ Next(C_1) then return 1 else return 0
else // k > 0
    foreach C ∈ Conf[S(n)] do
        if Reach(C_1, C, k − 1) = 1 ∧ Reach(C, C_2, k − 1) = 1 then
            return 1
        endif
    endfor
    return 0
endif
```

Proof. Let M be a S-space bounded NTM. Then there exists a constant d such that M reaches at most $2^{dS(n)}$ different configurations on inputs of length n. If M has an accepting computation on x, then there is one that is reachable in at most $2^{d \cdot S(n)}$ steps. Furthermore, every configuration of M on x can be expressed by a word of length $c \cdot S(n)$, where c is a constant. Here, we use that $S(n) \geq \log n$.

Again, the idea is to search the configuration graph for reachable accepting configurations. Unlike in the previous argument, we cannot explicitly construct the whole configuration graph or maintain a complete list of reachable positions. However, we can solve the problem by an on-the-fly construction of $G[M, T(n)]$. We define a recursive, deterministic procedure **Reach**(C_1, C_2, k), see Algorithm 2.2, that, given two configurations $C_1, C_2 \in \text{Conf}[S(n)]$ and an integer $k \in \mathbb{N}$, computes the following output:

$$\textbf{Reach}(C_1, C_2, k) = \begin{cases} 1 & \text{if } M \text{ can reach configuration } C_2 \text{ from } C_1 \\ & \text{in less than } 2^k \text{ steps;} \\ 0 & \text{otherwise.} \end{cases}$$

Let $f(n,k) = \max\{\text{space}_{\text{Reach}(C_1,C_2,k)} : C_1, C_2 \in \text{Conf}[S(n)]\}$. We have

- $f(n,0) = 0$
- $f(n,k+1) \leq c \cdot S(n) + f(n,k)$, where $c \cdot S(n)$ is the space used to write C (space constructibility of S)

Algorithm 2.3. M_{det}

Input: x
foreach *accepting configuration* $C_a \in Conf[S(n)]$ **do**
  if $Reach(C_0(x), C_a, d \cdot S(n)) = 1$ **then** accept
endfor
reject

Therefore, $f(n,k) \leq k \cdot c \cdot S(n)$. $L(M)$ can be decided by Algorithm 2.3. Since $\text{space}_{M_{\text{det}}}(x) = O(S(n)) + f(n, d \cdot S(n)) = O(S^2(n))$, we conclude that $L(M) \in \text{DSPACE}(O(S^2)) = \text{DSPACE}(S^2)$. Q.E.D.

Corollary 2.9.

(i) NLOGSPACE $\subseteq$ P.
(ii) NPSPACE $=$ PSPACE.
(iii) NP $\subseteq$ PSPACE.

Proof.

(i) $\text{NLOGSPACE} := \text{NSPACE}(O(\log n)) \subseteq \text{DTIME}(2^{O(\log n)})$
$= \text{DTIME}(n^{O(1)}) = \text{P}$.
(ii) $\text{NPSPACE} := \bigcup_{d \in \mathbb{N}} \text{NSPACE}(n^d) \subseteq \bigcup_{d \in \mathbb{N}} \text{DSPACE}(n^{2d}) = \text{PSPACE}$.
(iii) $\text{NP} := \bigcup_{d \in \mathbb{N}} \text{NTIME}(n^d) \subseteq \text{NPSPACE} = \text{PSPACE}$. Q.E.D.

2.3 The Theorem of Immerman and Szelepcsényi

Definition 2.10. Let $\mathcal{C}$ be a class of languages (e.g., a complexity class). Then, we define the class $\text{co}\mathcal{C} := \{\overline{L} : L \in \mathcal{C}\}$, where $\overline{L}$ is denotes the complement of L.

The deterministic complexity classes $\text{DTIME}(T)$ and $\text{DSPACE}(T)$ are obviously closed under the following operations:

- Union: $L, L' \in \mathcal{C} \implies L \cup L' \in \mathcal{C}$;
- Intersection: $L, L' \in \mathcal{C} \implies L \cap L' \in \mathcal{C}$;
- Complement: $L \in \mathcal{C} \implies \overline{L} \in \mathcal{C}$, i.e., $\mathcal{C} = \text{co}\mathcal{C}$.

The nondeterministic complexity classes $\text{NTIME}(T)$ and $\text{NSPACE}(S)$ are also closed under union and intersection. However, the closure under complement is not obvious and possibly incorrect in many instances. Actually, it is conjectured that the complexity class $\text{NTIME}(T)$ is not closed under complement. For $\text{NSPACE}(S)$, this conjecture had been standing for a long time when Immerman and Szelepcsényi presented the following surprising result in 1988.

Theorem 2.11 (Immerman und Szelepcsényi)**.**

$$\text{NSPACE}(S) = \text{coNSPACE}(S)$$

for any space constructible function $S(n) \geq \log n$.

The main idea of the proof is to "count inductively" all reachable configurations. Once the number $R_x(t)$ of configurations that can be reached in t steps is known, we can decide for every configuration C whether it is reachable in $t+1$ steps. If so, this can be verified by guessing an appropriate computation for C. Otherwise, we can verify that $C \notin \text{Next}(D)$ for all $R_x(t)$ configurations of D that are reachable in t steps. More generally, for a nondeterministic decision procedure of L, we only require that $x \in L$ iff there is an accepting computation of M on x. In particular, there can be rejecting computations on x although $x \in L$. To sharpen our terminology accordingly, we introducing the notion of an *error-free* nondeterministic computation or decision procedure.

Definition 2.12. An *error-free* nondeterministic computation procedure for a function f is a nondeterministic Turing machine M with the following properties:

(i) every computation of M on x stops with output either $f(x)$ or ? ("I don't know");

(ii) at least one computation of M produces the result $f(x)$.

An error-free nondeterministic decision procedure for a language L is an error-free nondeterministic computation procedure for its characteristic function χ_L.

We will now prove the following theorem which implies Theorem 2.11.

Theorem 2.13. Let $S(n) \geq \log n$ be space constructible. Then, for every $L \in \text{NSPACE}(S)$ there is an error-free S-space bounded decision procedure.

In particular, this implies that such a decision procedure also exists for $\overline{L}$ and, consequently, $\overline{L} \in \text{NSPACE}(S)$.

Proof. Let M be a S-space bounded NTM that decides L, $C_0(x)$ the initial configuration of M on x and $\text{Conf}[S(n)]$ the set of configurations of M with space usage $\leq S(n)$. As $S(n) \geq \log n$, every configuration $C \in \text{Conf}[S(n)]$ can be described by a word of length $S(n)$. Let

$$\text{Reach}_x(t) := \{C \in \text{Conf}[S(n)] : C \text{ is reachable from } C_0(x) \text{ in} \leq t \text{ steps}\}$$

and set $R_x(t) := |\text{Reach}_x(t)|$.

(1) There is a nondeterministic procedure M_0 with input x, r, t, C, where x is the input of M, $r, t \in \mathbb{N}$ and $C \in \text{Conf}[S(n)]$ $(n = |x|)$, such that if $r = R_x(t)$, then M_0 decides error-free in space $O(S(n))$ whether $C \in$

Algorithm 2.4. $M_0(x, r, t, C)$

```
m := 0
foreach D ∈ Conf[S(n)] do
    /* simulate (nondeterministically) at most t steps of M on x */
    C' = C_0(x)
    for t times do
        if C' ≠ D then
            guess C'' ∈ Next(C')
            C' := C''
        endif
    endfor
    if C' = D then                                  /* D was reached */
        m = m + 1
        if C ∈ Next(D) then output 1
    endif
endfor
if m = r then output 0 else output ?
```

Algorithm 2.5. M_1

Input: x
$r := 1$
for $t = 0$ **to** $t(|x|)$ **do**
  $m := 0$
  foreach $C \in Conf[S(n)]$ **do**
    $z := M_0(x, r, t, C)$ `/* Call of nondet. procedure` M_0 `*/`
    if $z = 1$ **then** $m := m + 1$
    if $z = ?$ **then output** ?
  endfor
  $r := m$
endfor
output r

$\text{Reach}_x(t+1)$. It does not matter how M_0 operates on (x, r, t, C) with $r \neq R_x(t)$.

Remark. The nondeterministic simulation of at most t steps, for $t = 2^{O(S(n))}$, can be done in space $O(S(n))$, e.g., by guessing a path step by step.

Let $r = R_x(t)$. We obtain:

- If $C \in \text{Reach}_x(t+1)$, there is a computation with output 1. Furthermore, there is no computation with output 0 since no computation passes through all configurations within $t+1$ steps without reaching C at least in the $(t+1)$st step.
- If $C \notin \text{Reach}_x(t+1)$, there is a computation of M_0 that outputs 0. This is the one that follows all computation paths of length at most t, checking for every configuration D met on such a path whether $D \notin \text{Next}(C)$. Moreover, no computation returns 1.

(2) Clearly, there is a function $t(n) = 2^{O(S(n))}$ such that M either halts after $t(n)$ steps or it enters a loop.

Lemma 2.14. There is an error-free nondeterministic $O(S(n))$-space bounded computation procedure for the function $x \mapsto R_x(t(|x|))$.

Proof. Algorithm 2.5 describes the procedure M_1 which calls the nondeterministic procedure M_0 (usually several times) and is therefore

nondeterministic itself. Each time $M_0(x, r, t, C)$ is called by M_1, we have $r = R_x(t)$ for the current values of r and t because:

- $t = 0 : r = 1 = R_x(0)$
- $t > 0 : r = |\{C :$ there is a computation of M_0 on input $(x, R_x(t-1), t-1, C)$ with output $1\}| = R_x(t)$.

In particular, the value of r at the end of a successful computation of M_1 equals $R_x(t(|x|))$. Since there is a computation of M_0 on (x, r, t, C) for all r, t with $r = R_x(t)$ that results in ?, there is also a computation of M_1 that computes the number $R_x(t(|x|))$. This proves the lemma. Q.E.D.

(3) Finally, Algorithm 2.6 specifies an error-free nondeterministic decision procedure for $L = L(M)$.

- Let $x \in L$. Hence, there is a computation of M_1 that results in $r = R_x(t(|x|))$. Then there exists an accepting configuration $C_a \in \text{Reach}_x(t(|x|))$ and therefore a computation of M_0 on $(x, r, t(|x|), C_a)$ with output 1. Therefore, there is a computation of $\tilde{M}$ with output "$x \in L$". On the other hand, it is clear that the answer "$x \in L$" is produced only if there is an accepting configuration C_a with $C_0(x) \vdash_M^x C_a$, that is, if indeed $x \in L$. We have thus shown: $x \in L$ iff there is a computation of $\tilde{M}$ with answer "$x \in L$".

Algorithm 2.6. $\tilde{M}$

Input: x
$r := M_1(x)$ `/* Call of` M_1 `*/`
if $r =$? **then output** ? **else**
  foreach *accepting* $C_a \in ConfIS(n)I$ **do**
    $z := M_0(x, r, t(|x|), C_a)$
    if $z = 1$ **then output** "$x \in L$"
    if $z =$? **then output** ?
  endfor
endif
output "$x \notin L$"

- Let $x \notin L$: Again, there is a computation of M_1 resulting in $r = R_x(t(|x|))$. As no accepting configuration C_a is reachable from $C_0(x)$, for every C_a there is a computation of M_0 on $(x, r, t(|x|), C_a)$ resulting in 0. Therefore, there is a computation of $\tilde{M}$ with answer "$x \notin L$". On the other hand, this answer is given only if M_0 has returned 0 for each C_a, that is, if no C_a is reachable from $C_0(x)$ or, in other words, if $x \notin L$.

Thus, we have shown that $\tilde{M}$ is an error-free nondeterministic decision procedure for $L = L(M)$ and therefore also for $\overline{L}$. Obviously, $\tilde{M}$ is $O(S(n))$-space bounded. By the Space Compression Theorem (Theorem 1.18), we obtain $\overline{L} \in \text{NSPACE}(S)$. Q.E.D.

In particular, it follows that coNLOGSPACE = NLOGSPACE.

3 Completeness

3.1 Reductions

Definition 3.1. Let $A \subseteq \Sigma^*, B \subseteq \Gamma^*$ be two languages. A function $f : \Sigma^* \to \Gamma^*$ is called a *reduction from A to B* if, for all $x \in \Sigma^*$, $x \in A \Leftrightarrow f(x) \in B$. To put it differently: If $f(A) \subseteq B$ and $f(\bar{A}) = f(\Sigma^* \setminus A) \subseteq (\Gamma^* \setminus B) = \bar{B}$. Hence, a reduction from A to B is also a reduction from $\bar{A}$ to $\bar{B}$.

Let $\mathcal{C}$ be a complexity class (of decision problems). A class of functions $\mathcal{F}$ provides an appropriate notion of *reducibility* for $\mathcal{C}$ if

- $\mathcal{F}$ is closed under composition, i.e.,

  $$\begin{aligned} &\text{if} \quad f : \Sigma^* \to \Gamma^* \in \mathcal{F} \\ &\text{and} \quad g : \Gamma^* \to \Delta^* \in \mathcal{F}, \\ &\text{then} \quad g \circ f : \Sigma^* \to \Delta^* \in \mathcal{F}. \end{aligned}$$

- $\mathcal{C}$ is closed under $\mathcal{F}$: If $B \in \mathcal{C}$ and $f \in \mathcal{F}$ is a reduction from A to B, then $A \in \mathcal{C}$.

For two problems A, B we say that A is $\mathcal{F}$-deducible to B if there is a function $f \in \mathcal{F}$ that is a reduction from A to B.

Notation: $A \leq_{\mathcal{F}} B$.

Definition 3.2. A problem B is *$\mathcal{C}$-hard under $\mathcal{F}$* if all problems $A \in \mathcal{C}$ are $\mathcal{F}$-reducible to B ($A \in \mathcal{C} \Rightarrow A \leq_{\mathcal{F}} B$).

A problem B is *$\mathcal{C}$-complete (under $\mathcal{F}$)* if $B \in \mathcal{C}$ and B is $\mathcal{C}$-hard (under $\mathcal{F}$).

The most important notions of reducibility in complexity theory are

- $\leq_p$: polynomial-time reducibility (given by the class of all polynomial-time computable functions)
- $\leq_{\log}$: log-space reducibility (given by the class of functions computable with logarithmic space)

Closure under composition for polynomial-time reductions is easy to show. If

$$f : \Sigma^* \to \Gamma^* \text{ is computable in time } O(n^k) \text{ by } M_f \text{ and}$$
$$g : \Gamma^* \to \Delta^* \text{ is computable in time } O(n^m) \text{ by } M_g,$$

then there are constants c, d such that $g \circ f : \Sigma^* \to \Delta^*$ is computable in time $c \cdot n^k + d(c \cdot n^k)^m = O(n^{k+m})$ by a machine that writes the output of M_f (whose length is bounded by $c \cdot n^k$) to a working tape and use it as the input for M_g.

In case of log-space reductions this trivial composition does not work since $f(x)$ can have polynomial length in $|x|$ and hence cannot be completely written to the logarithmically bounded work tape. However, we can use a modified machine M'_f that computes, for an input x and a position i, the i-th symbol of the output $f(x)$. Thus, $g(f(x))$ can be computed by simulating M_g, such that whenever it accesses the i-th symbol of the input, M'_f is called to compute it. The computation of M'_f on (x, i) can be done in logarithmic space (space needed for computation and for the counter i: $\log(n^k)$) the symbol $f(x, i)$ written to the tape needs only constant space. Furthermore, the computation of M_g only needs space logarithmic in the input length as $c \cdot \log(|f(x)|) = c \cdot \log(|x|^k) = c \cdot k \cdot \log(|x|) = O(\log(|x|))$.

3.2 NP-complete problems: SAT and variants

NP can be defined as the class of problems decidable in nondeterministic polynomial time:

Definition 3.3. $\text{NP} = \bigcup_{d \in \mathbb{N}} \text{NTIME}(n^d)$.

A different, in some sense more instructive, definition of NP is the class of problems with polynomially-time verifiable solutions:

Definition 3.4. $A \in \mathrm{NP}$ if, and only if, there is a problem $B \in \mathrm{P}$ and a polynomial p such that $A = \{x : \exists y(|y| \leq p(|x|) \wedge x\#y \in B)\}$.

The two definitions coincide: If A has polynomially verifiable solutions via $B \in \mathrm{P}$ and a polynomial p, then the following algorithm decides A in nondeterministic polynomial time:

Input: x
guess y with $|y| < p(n)$
check whether $x\#y \in B$
if *answer is yes* **then** accept **else** reject

Conversely, let $A \in \textsc{Ntime}(p(n))$, and M be a p-time bounded NTM that decides A. A computation of M on some input of length n is a sequence of at most $p(n)$ configurations of length $\leq p(n)$. Therefore, a computation of M can be described by a $p(n) \times p(n)$ table with entries from $Q \times \Sigma \cup \Sigma$ and thus by a word of length $p^2(n)$. Set

$$B = \{x\#y : y \text{ accepting computation of } M \text{ on } x\}.$$

We can easily see that $B \in \mathrm{P}$, and $x \in L$ if, and only if, there exists y with $|y| \leq p^2(n)$ such that $x\#y \in B$. Therefore, $L \in \mathrm{NP}$.

Theorem 3.5.

(i) $\mathrm{P} \subseteq \mathrm{NP}$.
(ii) $A \leq_{\mathrm{p}} B, B \in \mathrm{NP} \Rightarrow A \in \mathrm{NP}$.

Clearly NP is closed under polynomial-time reductions:

$$B \in \mathrm{NP}, A \leq_{\mathrm{p}} B \quad \Longrightarrow \quad A \in \mathrm{NP}.$$

B is NP-complete if

(1) $B \in \mathrm{NP}$ and
(2) $A \leq_{\mathrm{p}} B$ for all $A \in \mathrm{NP}$.

The most important open problem in complexity theory is **Cook's hypothesis**: $\mathrm{P} \neq \mathrm{NP}$.

For every NP-complete problem B we have:

$$P \neq \mathrm{NP} \quad \Longleftrightarrow \quad B \notin \mathrm{P}.$$

We recall the basics of propositional logic. Let $\tau = \{X_i : i \in \mathbb{N}\}$ be a finite set of propositional variables. The set AL of *propositional logic formulae* is defined inductively:

(1) $0, 1 \in \mathrm{PL}$ (the Boolean constants are formulae).
(2) $\tau \subseteq \mathrm{PL}$ (every propositional variable is a formula).
(3) If $\psi, \varphi \in \mathrm{PL}$, then also $\neg\psi$, $(\psi \wedge \varphi)$, $(\psi \vee \varphi)$ and $(\psi \to \varphi)$ are formulae in PL.

A *(propositional) interpretation* is a map $\mathfrak{I} : \sigma \to \{0,1\}$ for some $\sigma \subseteq \tau$. It is *suitable* for a formula $\psi \in \mathrm{PL}$ if $\tau(\psi) \subseteq \sigma$. Every interpretation $\mathfrak{I}$ that is suitable to ψ defines a logical value $[\![\psi]\!]^{\mathfrak{I}} \in \{0,1\}$ with the following definitions:

(1) $[\![0]\!]^{\mathfrak{I}} := 0$, $[\![1]\!]^{\mathfrak{I}} := 1$.
(2) $[\![X]\!]^{\mathfrak{I}} := \mathfrak{I}(X)$ for $X \in \sigma$.
(3) $[\![\neg\psi]\!]^{\mathfrak{I}} := 1 - [\![\psi]\!]^{\mathfrak{I}}$.
(4) $[\![\psi \wedge \varphi]\!]^{\mathfrak{I}} := \min([\![\psi]\!]^{\mathfrak{I}}, [\![\varphi]\!]^{\mathfrak{I}})$.
(5) $[\![\psi \vee \varphi]\!]^{\mathfrak{I}} := \max([\![\psi]\!]^{\mathfrak{I}}, [\![\varphi]\!]^{\mathfrak{I}})$.
(6) $[\![\psi \to \varphi]\!]^{\mathfrak{I}} := [\![\neg\psi \vee \varphi]\!]^{\mathfrak{I}}$.

A *model* of a formula $\psi \in \mathrm{PL}$ is an interpretation $\mathfrak{I}$ with $[\![\psi]\!]^{\mathfrak{I}} = 1$. Instead of $[\![\psi]\!]^{\mathfrak{I}} = 1$, we will write $\mathfrak{I} \models \psi$ and say $\mathfrak{I}$ *satisfies* ψ. A formula ψ is called *satisfiable* if a model for ψ exists. A formula ψ is called a *tautology* if every suitable interpretation for ψ is a model of ψ.

A formula ψ is obviously satisfiable iff $\neg\psi$ is not a tautology. Two formulae ψ and φ are called equivalent ($\psi \equiv \varphi$) if, for each $\mathfrak{I} : \tau(\psi) \cup \tau(\varphi) \to \{0,1\}$, we have $[\![\psi]\!]^{\mathfrak{I}} = [\![\varphi]\!]^{\mathfrak{I}}$. A formula φ follows from ψ (short, $\psi \models \varphi$) if, for every interpretation $\mathfrak{I} : \tau(\psi) \cup \tau(\varphi) \to \{0,1\}$ with $\mathfrak{I}(\psi) = 1$, $\mathfrak{I}(\varphi) = 1$ holds as well.

Comments. Usually, we omit unnecessary parentheses. As $\wedge$ and $\vee$ are semantically associative, we can use the following notations for conjunctions and disjunctions over $\{\psi_i \ : \ i \in I\}$: $\bigwedge_{i \in I} \psi_i$ respectively $\bigvee_{i \in I} \psi_i$. We fix the set of variables $\tau = \{X_i \ : \ i \in \mathbb{N}\}$ and encode X_i

by $X(\text{bin } i)$, i.e., a symbol X followed by the binary representation of the index i. This enables us to encode propositional logic formulae as words over a finite alphabet $\Sigma = \{X, 0, 1, \wedge, \vee, \neg, (,)\}$.

Definition 3.6. $\textsc{sat} := \{\psi \in \text{PL} : \psi \text{ is satisfiable}\}$.

Theorem 3.7 (Cook, Levin)**.** SAT is NP-complete.

Proof. It is clear that SAT is in NP because

$$\{\psi\#\mathfrak{I} \mid \mathfrak{I} : \tau(\psi) \to \{0,1\}, \mathfrak{I} \models \psi\} \in \text{P}.$$

Let A be some problem contained NP. We show that $A \leq_{\text{p}} \textsc{sat}$. Let $M = (Q, \Sigma, q_0, F, \delta)$ be a nondeterministic 1-tape Turing machine deciding A in polynomial time $p(n)$ with $F = F^+ \cup F^-$. We assume that every computation of M ends in either an accepting or rejecting final configuration, i.e., C is a final configuration iff $\text{Next}(C) = \varnothing$. Let $w = w_0 \cdots w_{n-1}$ be some input for M. We build a formula $\psi_w \in \text{PL}$ that is satisfiable iff M accepts the input w.

Towards this, let ψ_w contain the following propositional variables:

- $X_{q,t}$ for $q \in Q, 0 \leq t \leq p(n)$,
- $Y_{a,i,t}$ for $a \in \Sigma, 0 \leq i, t \leq p(n)$,
- $Z_{i,t}$ for $0 \leq i, t \leq p(n)$,

with the following intended meaning:

- $X_{q,t}$: "at time t, M is in state q,"
- $Y_{a,i,t}$: "at time t, the symbol a is written on field i,"
- $Z_{i,t}$: "at time t, M is at position i."

Finally,

$$\psi_w := \textsc{start} \wedge \textsc{compute} \wedge \textsc{end}$$

with

$$\textsc{start} := X_{q_0,0} \wedge \bigwedge_{i=0}^{n-1} Y_{w_i,i,0} \wedge \bigwedge_{i=n}^{p(n)} Y_{\square,i,0} \wedge Z_{0,0}$$

$$\textsc{compute} := \textsc{nochange} \wedge \textsc{change}$$

$$\textsc{nochange} := \bigwedge_{t<p(n),a\in\Sigma,i\neq j} (Z_{i,t} \wedge Y_{a,j,t} \rightarrow Y_{a,j,t+1})$$

$$\textsc{change} := \bigwedge_{t<p(n),i,a,q} \Big((X_{q,t} \wedge Y_{a,i,t} \wedge Z_{i,t}) \rightarrow \bigvee_{\substack{(q',b,m)\in\delta(q,a)\\ 0\leq i+m\leq p(n)}} (X_{q',t+1} \wedge Y_{b,i,t+1} \wedge Z_{i+m,t+1}) \Big)$$

$$\textsc{end} := \bigwedge_{t\leq p(n),q\in F^-} \neg X_{q,t}$$

Here, START "encodes" the input configuration at time 0. NOCHANGE ensures that no changes are made to the field at the current position. CHANGE represents the transition function.

It is straightforward to see that the map $w \mapsto \psi_w$ is computable in polynomial time.

(1) Let $w \in L(M)$. Every computation of M induces an interpretation of the propositional variables $X_{q,t}, Y_{a,i,t}, Z_{i,t}$. An accepting computation of M on w induces an interpretation that satisfies ψ_w. Therefore, $\psi_w \in$ SAT.

(2) Let $C = (q, y, p)$ be some configuration of M, $t \leq p(n)$. Set

$$\textsc{conf}[C,t] := X_{q,t} \wedge \bigwedge_{i=0}^{p(n)} Y_{y_i,i,t} \wedge Z_{p,t}.$$

Please note that START $= \textsc{conf}[C_0(w), 0]$. Thus,

$$\psi_w \models \textsc{conf}[C_0(w), 0]$$

holds. For every non-final configuration C of M and all $t < p(n)$, we obtain (because of the subformula COMPUTE of ψ_w) :

$$\psi_w \wedge \textsc{conf}[C,t] \models \bigvee_{C'\in \mathrm{Next}(C)} \textsc{conf}[C', t+1].$$

(3) Let $\mathfrak{I}(\psi_w) = 1$. From (1) and (2) it follows that there is at least one computation $C_0(w) = C_0, C_1, \ldots, C_r = C_{\mathrm{end}}$ of M on w with

$r \leq p(n)$ such that $\mathfrak{I}(\textsc{conf}[C_t, t]) = 1$ for each $t = 0, \ldots, v$. Furthermore, $\psi_w \models \neg\textsc{conf}[C, t]$ holds for all rejecting final configurations C of M and all t because of the subformula END of ψ_w. Therefore, C_{end} is accepting, and M accepts the input w.

We have thus shown that $\psi_w \in \textsc{sat}$ if, and only if, $w \in A$. Q.E.D.

Remark. The reduction $w \mapsto \psi_w$ is particularly easy; it is computable with *logarithmic space*.

A consequence from Theorem 3.7 is that SAT is NP-complete via LOGSPACE-reductions.

Even though SAT is NP-complete, the satisfiability problem may still be polynomially solvable for some interesting formulae classes $S \subseteq \text{PL}$. We show that for certain classes $S \subseteq \text{PL}$, $S \cap \textsc{sat} \in \text{P}$ while in other cases $S \cap \textsc{sat}$ is NP-complete.

Reminder. A *literal* is a propositional variable or its negation. A formula $\psi \in \text{PL}$ is in *disjunctive normal form (DNF)* if it is of the form $\psi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} Y_{ij}$, where Y_{ij} are literals. A formula ψ is in *conjunctive normal form (CNF)* if it has the form $\psi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} Y_{ij}$. A disjunction $\bigvee_j Y_{ij}$ is also called *clause*. Every formula $\psi \in \text{PL}$ is equivalent to a formula ψ_D in DNF and to a formula ψ_C in CNF.

$$\psi \equiv \psi_D := \bigvee_{\substack{\mathfrak{I}:\tau(\psi)\to\{0,1\} \\ \mathfrak{I}(\psi)=1}} \bigwedge_{X\in\tau(\psi)} X^{\mathfrak{I}}$$

with

$$X^{\mathfrak{I}} = \begin{cases} X & \text{if } \mathfrak{I}(X) = 1 \\ \neg X & \text{if } \mathfrak{I}(X) = 0\,, \end{cases}$$

and analogously for CNF.

The translations $\psi \mapsto \psi_D, \psi \mapsto \psi_C$ are computable but generally not in polynomial time. The formulae ψ_D and ψ_C can be exponentially longer than ψ as there are $2^{|\tau(\psi)|}$ possible maps $\mathfrak{I} : \tau(\psi) \to \{0, 1\}$.

$\textsc{sat-dnf} := \{\psi \text{ in DNF } : \ \psi \text{ satisfiable}\}$ and

$$\text{SAT-CNF} := \{\psi \text{ in CNF } : \ \psi \text{ satisfiable}\}$$

denote the set of all satisfiable formulae in DNF and CNF, respectively.

Theorem 3.8. SAT-DNF $\in$ LOGSPACE $\subseteq$ P.

Proof. $\psi = \bigvee_i \bigwedge_{j=1}^{m_i} Y_{ij}$ is satisfiable iff there is an i such that no variable in $\{Y_{ij} : j = 1, \ldots, m_i\}$ occurs both positively and negatively. Q.E.D.

Theorem 3.9. SAT-CNF is NP-complete via LOGSPACE-reduction.

Proof. The proof follows from the one of Theorem 3.7. Consider the formula

$$\psi_w = \text{START} \wedge \text{COMPUTE} \wedge \text{END}\,.$$

From the proof, we see that START and END are already in CNF. The same is true for the subformula NOCHANGE of COMPUTE, only CHANGE is left. CHANGE is a conjunction of formulae that have the form

$$\alpha : X \wedge Y \wedge Z \to \bigvee_{j=1}^{r} X_j \wedge Y_j \wedge Z_j.$$

Here, $r \leq \max_{(q,a)} |\delta(q,a)|$ is fixed, i.e., independent of n and w. But we have

$$\alpha \equiv (X \wedge Y \wedge Z \to \bigvee_{j=1}^{r} U_j) \wedge \bigwedge_{j=1}^{r} (U_j \to X_j) \wedge (U_j \to Y_j) \wedge (U_j \to Z_j)).$$

Therefore, $A \leq_{\log}$ SAT-CNF for each $A \in$ NP. Q.E.D.

3.3 P-complete problems

A (propositional) *Horn formula* is a formula $\psi = \bigwedge_i \bigvee_j Y_{i_j}$ in CNF where every disjunction $\bigvee_j Y_j$ contains at most one positive literal. Horn formulae can also be written as implications by the following equivalences:

$$\neg X_1 \vee \cdots \vee \neg X_k \vee X \equiv (X_1 \wedge \cdots \wedge X_k) \to X,$$
$$\neg X_1 \vee \cdots \vee \neg X_k \equiv (X_1 \wedge \cdots \wedge X_k) \to 0.$$

Let HORN-SAT $= \{\psi \in \text{PL} \ : \ \psi \text{ a satisfiable Horn formula}\}$. We know from mathematical logic:

Theorem 3.10. HORN-SAT $\in$ P.

This follows, e.g., by unit resolution or the marking algorithm.

Theorem 3.11. HORN-SAT is P-complete via logspace reduction.

Proof. Let $A \in P$ and M a deterministic 1-tape Turing machine, that decides A in time $p(n)$. Looking at the reduction $w \mapsto \psi_w$ from the proof of Theorem 3.7, we see that the formulae START, NOCHANGE and END are already Horn formulae. Since M was chosen to be deterministic, i.e., $|\delta(q,a)| = 1$, CHANGE takes the form $(X \wedge Y \wedge Z) \to (X' \wedge Y' \wedge Z')$. This is equivalent to the Horn formula $(X \wedge Y \wedge Z) \to X' \wedge (X \wedge Y \wedge Z) \to Y' \wedge (X \wedge Y \wedge Z) \to Z'$. We thus have a logspace computable function $w \mapsto \widehat{\psi}_w$ such that

- $\widehat{\psi}_w$ is a Horn formula,
- M accepts w iff $\widehat{\psi}_w$ is satisfiable.

Therefore, $A \leq_{\log}$ HORN-SAT. Q.E.D.

Another fundamental P-complete problem is the computation of winning regions in finite (reachability) games.

Such a game is given by a game graph $G = (V, V_0, V_1, E)$ with a finite set V of positions, partitioned into V_0 and V_1, such that Player 0 moves from positions $v \in V_0$, moves from positions $v \in V_1$. All moves are along edges, and we use the term *play* to describe a (finite or infinite) sequence $v_0 v_1 v_2 \ldots$ with $(v_i, v_{i+1}) \in E$ for all i. We use a simple positional winning condition: Move or lose! Player σ wins at position v if $v \in V_{1-\sigma}$ and $vE = \varnothing$, i.e., if the position belongs to the opponent and there are no possible moves possible from that position. Note that this winning condition only applies to finite plays, infinite plays are considered to be a draw.

A *strategy* for Player σ is a mapping

$$f : \{v \in V_\sigma : vE \neq \varnothing\} \to V$$

with $(v, f(v)) \in E$ for all $v \in V$. We call f *winning* from position v if all plays that start at v and are consistent with f are won by Player σ.

We now can define *winning regions* W_0 and W_1:

$$W_\sigma = \{v \in V : \text{Player } \sigma \text{ has a winning strategy from position } v\}.$$

This leads to several algorithmic problems for a given game G: The computation of winning regions W_0 and W_1, the computation of winning strategies, and the associated decision problem

$$\textsc{game} := \{(G, v) : \text{Player 0 has a winning strategy for } G \text{ from } v\}.$$

Theorem 3.12. GAME is P-complete and decidable in time $O(|V| + |E|)$.

A simple polynomial-time approach to solve GAME is to compute the winning regions inductively: $W_\sigma = \bigcup_{n\in\mathbb{N}} W_\sigma^n$, where

$$W_\sigma^0 = \{v \in V_{1-\sigma} : vE = \varnothing\}$$

is the set of terminal positions which are winning for Player σ, and

$$W_\sigma^{n+1} = \{v \in V_\sigma : vE \cap W_\sigma^n \neq \varnothing\} \cup \{v \in V_{1-\sigma} : vE \subseteq W_\sigma^n\}$$

is the set of positions from which Player σ can win in at most $n+1$ moves.

After $n \leq |V|$ steps, we have that $W_\sigma^{n+1} = W_\sigma^n$, and we can stop the computation here.

To solve GAME in linear time, use the slightly more involved Algorithm 3.1. Procedure Propagate will be called once for every edge in the game graph, so the running time of this algorithm is linear with respect to the number of edges in $\mathcal{G}$.

The problem GAME is equivalent to the satisfiability problem for propositional Horn formulae. We recall that propositional Horn formulae are finite conjunctions $\bigwedge_{i\in I} C_i$ of clauses C_i of the form

$$X_1 \wedge \ldots \wedge X_n \rightarrow X \quad \text{or}$$
$$\underbrace{X_1 \wedge \ldots \wedge X_n}_{\text{body}(C_i)} \rightarrow \underbrace{0}_{\text{head}(C_i)}.$$

A clause of the form X or $1 \rightarrow X$ has an empty body.

Algorithm 3.1. A linear time algorithm for GAME

Input: A game $\mathcal{G} = (V, V_0, V_1, E)$
Output: Winning regions W_0 and W_1
foreach $v \in V$ **do** `/* 1: Initialisation */`
  $\text{win}[v] := \bot$
  $P[v] := \varnothing$
  $n[v] := 0$
endfor
foreach $(u, v) \in E$ **do** `/* 2: Calculate` P `and` n `*/`
  $P[v] := P[v] \cup \{u\}$
  $n[u] := n[u] + 1$
endfor
foreach $v \in V_0$ **do** `/* 3: Calculate win */`
  if $n[v] = 0$ **then** Propagate(v, 1)
endfor
foreach $v \in V \setminus V_0$ **do**
  if $n[v] = 0$ **then** Propagate(v, 0)
endfor
returnwin
procedure Propagate(v, σ)
if $\text{win}[v] \neq \bot$ **then** return
$\text{win}[v] := \sigma$ `/* 4: Mark` v `as winning for player` σ `*/`
foreach $u \in P[v]$ **do** `/* 5: Propagate change to predecessors */`
  $n[u] := n[u] - 1$ **if** $u \in V_\sigma$ *or* $n[u] = 0$ **then** Propagate(u, σ)
endfor

We will show that SAT-HORN and GAME are mutually reducible via logspace and linear-time reductions.

(1) GAME $\leq_{\text{log-lin}}$ SAT-HORN
For a game $\mathcal{G} = (V, V_0, V_1, E)$, we construct a Horn formula $\psi_\mathcal{G}$ with clauses

$$v \to u \qquad \text{for all } u \in V_0 \text{ and } (u, v) \in E, \text{ and}$$
$$v_1 \wedge \ldots \wedge v_m \to u \qquad \text{for all } u \in V_1 \text{ and } uE = \{v_1, \ldots, v_m\}.$$

The minimal model of $\psi_\mathcal{G}$ is precisely the winning region of Player 0, so

$$(\mathcal{G}, v) \in \text{GAME} \iff \psi_\mathcal{G} \wedge (v \to 0) \text{ is unsatisfiable.}$$

(2) SAT-HORN $\leq_{\text{log-lin}}$ GAME
For a Horn formula $\psi(X_1, \ldots, X_n) = \bigwedge_{i \in I} C_i$, we define a game $\mathcal{G}_\psi = (V, V_0, V_1, E)$ as follows:

$$V = \underbrace{\{0\} \cup \{X_1, \ldots, X_n\}}_{V_0} \cup \underbrace{\{C_i : i \in I\}}_{V_1} \text{ and}$$
$$E = \{X_j \to C_i : X_j = \text{head}(C_i)\} \cup \{C_i \to X_j : X_j \in \text{body}(C_i)\},$$

i.e., , Player 0 moves from a variable to some clause containing the variable as its head, and Player 1 moves from a clause to some variable in its body. Player 0 wins a play if, and only if, the play reaches a clause C with $\text{body}(C) = \varnothing$. Furthermore, Player 0 has a winning strategy from position X if, and only if, $\psi \models X$, so we have

$$\text{Player 0 wins from position } 0 \iff \psi \text{ is unsatisfiable.}$$

In particular, GAME is P-complete, and SAT-HORN is solvable in linear time.

3.4 NLogspace-complete problems

We already know that the reachability problem, i.e. to decide, given a directed graph G and two nodes a and b, whether there is a path from a to b in G, is in NLogspace.

Theorem 3.13. REACHABILITY is NLogspace-complete.

Proof. Let A be an arbitrary problem in NLogspace. There is a nondeterministic Turing machine M that decides A with workspace $c \log n$. We prove that $A \leq_{\log}$ REACHABILITY by associating, with every input x for M, a graph $G_x = (X_x, E_x)$ and two nodes a and b, such that M accepts x if, and only if, there is a path from a to b in G_x. The set of nodes of G_x is

$$V_x := \{C : C \text{ is a partial configuration of } M \text{ with workspace } c \log |x|\} \cup \{b\},$$

and the set of edges is

$$E_x := \{(C, C') : (C, x) \vdash_M (C'x)\} \cup \{(C_a, b) : C_a \text{ is accepting}\}.$$

Recall that a partial configuration is a configuration without the description of the input. Each partial configuration in V_x can be described by a word of length $O(\log |x|)$. Further we define a to be the initial partial configuration of M. Clearly (G_x, a, b) is constructible with logarithmic space from x and there is a path from a to b in G_x if, and only if, there is an accepting computation of M on x. Q.E.D.

We next discuss a variant of SAT that is NLOGSPACE-complete.

Definition 3.14. A formula is in r-CNF if it is in CNF and every clause contains at most r literals: $\psi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} Y_{i_j}$ with $m_i \leq r$ for all i. Furthermore, r-SAT $:= \{\psi \text{ in } r\text{-CNF} \ : \ \psi \text{ is satisfiable}\}$.

It is known that r-SAT is NP-complete for all $r \geq 3$.

To the contrary, 2-SAT can be solved in polynomial time, e.g., by resolution:

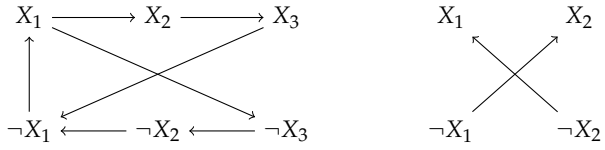
- The resolvent of two clauses with ≤ 2 literals contains at most 2 literals.
- At most $O(n^2)$ clauses with ≤ 2 literals can be formed with n variables.

Hence, we obtain that $\text{Res}^*(\psi)$ for a formula ψ in 2-CNF can be computed in polynomial time. One can show an even stronger result.

Theorem 3.15. 2-SAT is in NLOGSPACE.

Proof. We show that $\{\psi : \psi \text{ in 2-CNF}, \psi \text{ unsatisfiable}\} \in$ NLOGSPACE. Then, by the Theorem of Immerman and Szelepcsényi, also 2-SAT $\in$ NLOGSPACE. The reduction maps a formula $\psi \in$ 2-CNF to the following directed graph $G_\psi = (V, E)$:

- $V = \{X, \neg X : X \in \tau(\psi)\}$ represents the literals of ψ.
- $E = \{(Y, Z) : \psi \text{ contains a clause equivalent to } (Y \to Z)\}$.



(a) G_{ψ_a} (b) G_{ψ_b}

Figure 3.1. Graphs for $\psi_a = X_1 \wedge (\neg X_1 \vee X_2) \wedge (X_3 \vee \neg X_2) \wedge (\neg X_3 \vee \neg X_1)$ and $\psi_b = (X_1 \vee X_2)$

Example 3.16. Figures 3.1(a) and 3.1(b) show the graphs constructed for an unsatisfiable and a satisfiable 2-CNF formula, respectively.

Lemma 3.17 (Krom-Criterion)**.** Let ψ be in 2-CNF. ψ is unsatisfiable if, and only if, there exists a variable $X \in \tau(\psi)$ such that G_ψ contains a path from X to $\neg X$ and one from $\neg X$ to X.

The problem

$$L = \{(G, a, b) : G \text{ directed graph, there is a path from } a \text{ to } b\}$$

is also called the labyrinth problem. A formula ψ is unsatisfiable if, and only if, there exists a variable $X \in \tau(\psi)$ such that $(G_\psi, X, \neg X) \in L$ and $(G_\psi, \neg X, X) \in L$. Since $L \in$ NLogspace, the claim follows. Q.E.D.

Proof (of Lemma 3.17). We use the notation $Y \rightarrow_\psi^* Z$ to denote that there exists a path from Y to Z in G_ψ.

Let $\mathfrak{I}$ be an interpretation such that $\mathfrak{I}(\psi) = 1$. Then, $\mathfrak{I}(Y) = 1, Y \rightarrow_\psi^* Z \implies \mathfrak{I}(Z) = 1$. Hence, if $X \rightarrow_\psi^* \neg X \rightarrow_\psi^* X$, then ψ is unsatisfiable.

Conversely, for all $X \in \tau(\psi)$, either not $X \rightarrow_\psi^* \neg X$ or not $\neg X \rightarrow_\psi^* X$. In this case, Algorithm 3.2 constructs an interpretation $\mathfrak{I}$ such that $\mathfrak{I}(\psi) = 1$.

It is not possible to produce conflicting assignments resulting from $Y \rightarrow_\psi^* Z$ as well as $Y \rightarrow_\psi^* \neg Z$ since this would imply $\neg Z \rightarrow_\psi^* \neg Y$ and $Z \rightarrow_\psi^* \neg Y$, and hence $Y \rightarrow_\psi^* Z \rightarrow_\psi^* \neg Y$. But Y was chosen as to not have this property.

Algorithm 3.2

$U := \tau(\psi) \cup \neg\tau(\psi)$
while $U \neq \varnothing$ **do**
  choose $Y \in U$ such that $Y \to_\psi^* \neg Y$ does not hold
  $\mathfrak{I}(Y) := 1$
  $U := U - \{Y, \neg Y\}$
  foreach Z *such that* $Y \to_\psi^* Z$ **do**
    $\mathfrak{I}(Z) := 1$
    $U := U - \{Z, \neg Z\}$
  endfor
endwhile

Thus, Algorithm 3.2 constructs an interpretation $\mathfrak{I}$ since, for every variable $X \in \tau(\psi)$, either $\mathfrak{I}(X) = 1$ or $\mathfrak{I}(\neg X) = 1$. However, due to the nondeterministic choice of Y in each loop, the resulting interpretation is not uniquely determined.

Let $\mathfrak{I}$ be an interpretation constructed by Algorithm 3.2. It remains to prove that $\mathfrak{I}$ satisfies each clause $(Z \vee Z')$, and thus ψ.

Otherwise, there is a clause $(Z \vee Z')$ such that $\mathfrak{I}(Z) = \mathfrak{I}(Z') = 0$, i.e., $\mathfrak{I}(\neg Z) = 1$. This implies, that the algorithm has chosen a literal Y such that $Y \to_\psi^* \neg Z$ but $Y \to_\psi^* \neg Y$ does not hold. Since $\neg Z \to_\psi^* Z'$, we obtain $Y \to_\psi^* Z'$ and hence $\mathfrak{I}(Z') = 1$, which is a contradiction. Q.E.D.

Remark 3.18. Formulae in 2-CNF are sometimes called *Krom-formulae*.

Theorem 3.19. 2-SAT is NLOGSPACE-complete.

Proof. We prove that REACHABILITY $\leq_{\log}$ 2-SAT.

Given a directed graph $G = (V, E)$ with nodes a and b, we construct the 2-CNF formula

$$\psi_{G,a,b} := a \wedge \bigwedge_{(u,v)\in E} (u \to v) \wedge \neg b.$$

Clearly this defines a logspace-reduction from the reachability problem to 2-SAT. Q.E.D.

3.5 A Pspace-complete problem

Let us first recall two important properties of the complexity class $\textsc{Pspace} := \bigcup_k \textsc{Dspace}(n^k)$.

- $\textsc{Pspace} = \bigcup_{k\in\mathbb{N}} \textsc{Nspace}(n^k) = \textsc{NPspace}$ because, by the Theorem of Savitch, $\textsc{Nspace}(S) \subseteq \textsc{Dspace}(S^2)$.
- $\textsc{NP} \subseteq \textsc{Pspace}$ since $\textsc{Ntime}(n^k) \subseteq \textsc{Nspace}(n^k) \subseteq \textsc{Dspace}(n^{2k}) \subseteq \textsc{Pspace}$.

A problem A is Pspace-hard if $B \leq_{\mathrm{p}} A$ for all $B \in \textsc{Pspace}$. A is Pspace-complete if $A \in \textsc{Pspace}$ and A is Pspace-hard.

As an example of Pspace-complete problems, we consider the evaluation problem for quantified propositional formulae (also called QBF for "quantified Boolean formulae").

Definition 3.20. *Quantified propositional logic* is an extension of (plain) propositional logic. It is the smallest set closed under disjunction, conjunction and complement that allows quantification over propositional variables in the following sense: If ψ is a formula from quantified propositional logic and X a propositional variable, then $\exists X\psi, \forall X\psi$ are also quantified propositional formulae.

Example 3.21. $\exists X(\forall Y(X \vee Y) \wedge \exists Z(X \vee Z))$.

By $\mathrm{free}(\psi)$ we denote the set of free propositional variables in ψ. Every propositional interpretation $\mathfrak{I} : \sigma \to \{0,1\}$ with $\sigma \subseteq \tau$ defines logical values $\mathfrak{I}(\psi)$ for all quantified propositional formulae ψ with $\mathrm{free}(\psi) \subseteq \sigma$. Let $\mathfrak{I}$ be an interpretation and $X \in \tau$ a propositional variable. Further, we write $\mathfrak{I}[X = 1]$ for the interpretation that agrees with $\mathfrak{I}$ on all $Y \in \tau, Y \neq X$ and interprets X with 1. Analogously, let $\mathfrak{I}[X = 0]$ be the interpretation with $\mathfrak{I}[X/0](Y) = \mathfrak{I}(Y)$ for $Y \neq X$ and $\mathfrak{I}[X/0](X) = 0$. Then, $\mathfrak{I}(\exists X\psi) = 1$ if, and only if, $\mathfrak{I}[X/0](\psi) = 1$ or $\mathfrak{I}[X/1](\psi) = 1$. Similarly, $\mathfrak{I}(\forall X\psi) = 1$ if, and only if, $\mathfrak{I}[X/0](\psi) = 1$ and $\mathfrak{I}[X/1](\psi) = 1$.

Observe that if $\mathrm{free}(\psi) = \varnothing$ the value $\mathfrak{I}(\psi) \in \{0,1\}$ does not depend on a concrete interpretation $\mathfrak{I}$; we have either $\mathfrak{I}(X) = 1$ (ψ is *satisfied*) or $\mathfrak{I}(X) = 0$ (ψ is *unsatisfied*). The formula $\exists X(\forall Y(X \vee Y) \wedge \exists Z(X \vee Z))$ is satisfied, for example.

Algorithm 3.3. Eval$(\psi, \mathfrak{I})$

Input: $\psi, \mathfrak{I}$
if $\psi = X \in V$ **then** return $\mathfrak{I}(X)$
if $\psi = (\varphi_1 \vee \varphi_2)$ **then**
  if $Eval(\varphi_1, \mathfrak{I}) = 1$ **then** return 1 **else** return Eval$(\varphi_2, \mathfrak{I})$
endif
if $\psi = (\varphi_1 \wedge \varphi_2)$ **then**
  if $Eval(\varphi_1, \mathfrak{I}) = 0$ **then** return 0 **else** return Eval$(\varphi_2, \mathfrak{I})$
endif
if $\psi = \neg\varphi$ **then** return $1 -$ Eval$(\varphi, \mathfrak{I})$
if $\psi = \exists X\varphi$ **then**
  if $Eval(\varphi, \mathfrak{I}[X = 0]) = 1$ **then** return 1 **else**
    return Eval$(\varphi, \mathfrak{I}[X = 1])$
  endif
endif
if $\psi = \forall X\varphi$ **then**
  if $Eval(\varphi, \mathfrak{I}[X = 0]) = 0$ **then** return 0 **else**
    return Eval$(\varphi, \mathfrak{I}[X = 1])$
  endif
endif

Definition 3.22.

$$\text{QBF} := \{\psi \text{ a quantified PL formula} : \text{free}(\psi) = \varnothing\ ,\ \psi \text{ true}\}.$$

Remark 3.23. Let $\psi = \psi(X_1, \ldots, X_n)$ be a propositional formula (i.e., one that does not contain quantifiers). Then,

$$\psi \in \text{SAT} \iff \exists X_1 \ldots \exists X_n \psi \in \text{QBF}.$$

QBF is therefore at least as hard as SAT. Actually, we will show that QBF is PSPACE-complete.

Theorem 3.24. QBF $\in$ PSPACE.

Proof. The recursive procedure Eval$(\psi, \mathfrak{I})$ presented in Algorithm 3.3 computes the value $\mathfrak{I}(\psi)$ for a quantified propositional formula ψ and $\mathfrak{I}$: free$(\psi) \to \{0, 1\}$.

This procedure uses $O(n^2)$ space. It is easy to see that $\mathfrak{I}(\psi)$ is computed correctly. Q.E.D.

Theorem 3.25. QBF is PSPACE-hard.

Proof. Consider a problem A in PSPACE and let M be some n^k-space bounded 1-tape TM with $L(M) = A$. Every configuration of M on some input w of length n can be described by a tuple $\bar{X}$ of propositional variables consisting of:

$$\begin{array}{lll} X_q & (q \text{ is state of } M) & : \text{“}M \text{ is in state } q\text{”}, \\ X'_{a,i} & (a \text{ tape symbol}, i \leq n^k) & : \text{“symbol } a \text{ is on field } i\text{”}, \\ X''_j & (j \leq n^k) & : \text{“}M \text{ is on position } j\text{”}. \end{array}$$

As in the NP-completeness proof for SAT, we construct formulae $\textsc{conf}(\bar{X})$, $\textsc{next}(\bar{X}, \bar{Y})$, $\textsc{input}_w(\bar{X})$ and $\textsc{acc}(\bar{X})$ with the following intended meanings:

- $\textsc{conf}(\bar{X})$: $\bar{X}$ encodes some configuration, i.e., exactly one X_q is true, exactly one $X'_{a,i}$ is true for every i, and exactly one X''_j is true.
- $\textsc{input}_w(\bar{X})$: $\bar{X}$ encodes the initial configuration of M on $w = w_0 \dots w_{n-1}$:

$$\textsc{input}_w(\bar{X}) := \textsc{conf}(\bar{X}) \wedge X_{q_0} \wedge \bigwedge_{i=0}^{n-1} X'_{w_i,i} \wedge \bigwedge_{i=n}^{n^k} X'_{\square,i} \wedge X''_0.$$

- $\textsc{acc}(\bar{X})$: $\bar{X}$ is an accepting configuration:

$$\textsc{acc}(\bar{x}) := \textsc{conf}(\bar{X}) \wedge \bigvee_{q \in E^+} X_q.$$

- $\textsc{next}(\bar{X}, \bar{Y})$: $\bar{Y}$ is a successor configuration of $\bar{X}$:

$$\textsc{next}(\bar{X}, \bar{Y}) := \bigwedge_i \Big(X''_i \to \big(\bigwedge_{a,j \neq i} (Y'_{a,j} \leftrightarrow X_{a,j}) \wedge \bigwedge_{\substack{\delta(q,a)=(q',b,m) \\ 0 \leq m+i \leq n^k}} (X_q \wedge X'_{a,i} \to Y_{q'} \wedge Y'_{b,i} \wedge Y''_{i+m}) \big) \Big).$$

Given w, these formulae can be constructed in polynomial time.

Furthermore, we define the predicate

$$\textsc{eq}(\bar{X}, \bar{Y}) := \bigwedge_q (X_q \leftrightarrow Y_q) \wedge \bigwedge_{a,i} (X'_{a,i} \leftrightarrow Y'_{a,i}) \wedge \bigwedge_j (X''_j \leftrightarrow Y''_j).$$

We inductively construct formulae $\textsc{reach}_m(\bar{X}, \bar{Y})$ expressing that $\bar{X}$ and $\bar{Y}$ encode configurations and $\bar{Y}$ is accessible from $\bar{X}$ in at most 2^m steps. For $m = 0$, let

$$\textsc{reach}_0(\bar{X}, \bar{Y}) := \textsc{conf}(\bar{X}) \wedge \textsc{conf}(\bar{Y}) \wedge (\textsc{eq}(\bar{X}, \bar{Y}) \vee \textsc{next}(\bar{X}, \bar{Y})).$$

A naïve way to define $\textsc{reach}_{m+1}$ would be

$$\textsc{reach}_{m+1}(\bar{X}, \bar{Y}) := \exists \bar{Z} (\textsc{reach}_m(\bar{X}, \bar{Z}) \wedge \textsc{reach}_m(\bar{Z}, \bar{Y})).$$

But then $|\textsc{reach}_{m+1}| \geq 2 \cdot |\textsc{reach}_m|$ so that $|\textsc{reach}_m| \geq 2^m$ and hence grows exponentially. We can, however, construct $\textsc{reach}_{m+1}$ differently so that the exponential growth of the formula length is avoided by using universal quantifiers:

$$\textsc{reach}_{m+1}(\bar{X}, \bar{Y}) := \exists \bar{Z} \forall \bar{U} \forall \bar{V} \left(\begin{array}{c} (\textsc{eq}(\bar{X}, \bar{U}) \wedge \textsc{eq}(\bar{Z}, \bar{V})) \\ \vee (\textsc{eq}(\bar{Z}, \bar{U}) \wedge \textsc{eq}(\bar{Y}, \bar{V})) \end{array} \right) \rightarrow \textsc{reach}_m(\bar{U}, \bar{V}).$$

We now obtain:

$$|\textsc{reach}_0| = O(n^k) \text{ for some appropriate } k, \text{ and}$$
$$|\textsc{reach}_{m+1}| = |\textsc{reach}_m| + O(n^k).$$

Hence, $|\text{Reach}_m| = O(m \cdot n^k)$.

If M accepts the input w using space n^k it performs at most $\leq 2^{c \cdot n^k}$ steps for some constant c. Set $m := c \cdot n^k$ and

$$\psi_w := \exists \bar{X}\ \exists \bar{Y} (\textsc{input}(\bar{X}) \wedge \textsc{acc}(\bar{Y}) \wedge \textsc{reach}_m(\bar{X}, \bar{Y})).$$

Obviously, ψ_w is constructable from w in polynomial time and $\psi_w \in$ QBF if and only if $w \in L(M)$. Therefore, QBF is PSPACE-complete. Q.E.D.

4 Oracles and the polynomial hierarchy

4.1 Oracle Turing machines

Definition 4.1. A deterministic (respectively nondeterministic) *oracle Turing machine* is a Turing machine with a designated oracle tape and three special states ? (query), Y (yes) and N (no).

A configuration C of an Oracle Turing machine with k working tapes and a distinguished oracle tape is a tuple

$$C = (q, p_0, \cdots, p_k, w_0, \cdots, w_k),$$

where

- q is the state of the Turing machine,
- $p_0, \ldots, p_k$ are the head positions (p_0 is the head position of the oracle tape), and
- $w_0 \ldots, w_k$ are the head inscriptions (w_0 is the inscription of the oracle tape).

The computation (respectively the computation tree) of an oracle Turing machine depends on a previously defined oracle set $A \subseteq \Sigma^*$ (where Σ is the alphabet of M). The successor configurations of a configuration C are defined as usual for $q \neq ?$ while the successor configuration C' for $q = ?$ is defined as :

$$C' = \begin{cases} (\mathrm{Y}, 0, p_1, \ldots, p_k, \varepsilon, w_1, \ldots w_k) & \text{if } w_0 \in A \\ (\mathrm{N}, 0, p_1, \ldots, p_k, \varepsilon, w_1, \ldots, w_k) & \text{if } w_0 \notin A \end{cases}$$

where ε is the empty word. The oracle therefore determines whether or not w_0 (the inscription of the oracle tape) is in A. The machine consequently enters the corresponding state (Y or N) and the inscription of the oracle tape is erased.

Definition 4.2. Let M be an Oracle Turing machine and $A \subseteq \Sigma^*$ be some oracle set. Then the accepted language is

$$L(M^A) := \{x : M \text{ accepts the input } x \text{ with oracle } A\}.$$

Based on the oracle set A, we define the following complexity classes:

(i) $\mathrm{P}^A := \{L :$ there is a deterministic Oracle TM M that decides L using oracle A in polynomial time$\}$.

(ii) $\mathrm{NP}^A := \{L :$ there is a nondeterministic Oracle TM M that decides L using oracle A in polynomial time$\}$.

Let $\mathcal{C}$ be some class of languages, e.g., a complexity class. Then

$$\mathrm{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{P}^A \qquad \text{and} \qquad \mathrm{NP}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{NP}^A.$$

Example 4.3.

(a) Let $B \in \mathrm{NP}$. Then $B \in \mathrm{P}^{\mathrm{SAT}}$. Since SAT is NP-hard, there is a polynomially computable function f with $x \in B \iff f(x) \in \mathrm{SAT}$. The following oracle algorithm then decides B:

Input: x
Compute $f(x)$
Query the oracle whether $f(x) \in \mathrm{SAT}$
if Y **then** accept
if N **then** reject

(b) It is likely that $\mathrm{NP} \subsetneq \mathrm{P}^{\mathrm{SAT}}$ as every $B \in \mathrm{coNP}$ is in $\mathrm{P}^{\mathrm{SAT}}$. One can use the preceding algorithm and interchange the behaviour for the answers Yes and No.

(c) Let $B := \{(G,k) : G \text{ a graph}, \omega(G) = k\}$ where $\omega(G)$ is the maximal number of nodes of cliques in G. Reminder: The problem $\mathrm{CLIQUE} := \{(G,k) : k \leq \omega(G)\}$ is NP-complete. It is straightforward to see that $B \in P^{\mathrm{CLIQUE}}$:

Input: G, k
Query the oracle whether $(G, k) \in$ CLIQUE
if N **then** reject **else**
  Query the oracle whether $(G, k+1) \in$ CLIQUE
  if N **then** accept
  if Y **then** reject
endif

4.2 The polynomial hierarchy

Definition 4.4. We define the complexity classes Σ_k^p, Π_k^p, and Δ_k^p for all $k \in \mathbb{N}$:

- $\Sigma_0^p := \Pi_0^p := \Delta_0^p := \mathrm{P}$
- $\Sigma_{k+1}^p := \mathrm{NP}^{\Sigma_k^p}$
- $\Pi_k^p := \mathrm{co}\Sigma_k^p = \{\overline{A} : A \in \Sigma_k^p\}$
- $\Delta_{k+1}^p := \mathrm{P}^{\Sigma_k^p}$

Theorem 4.5. The classes Σ_k^p, Π_k^p, and Δ_k^p have the following elementary properties:

(i) $\Delta_1^p = \mathrm{P}$.

(ii) $\Sigma_1^p = \mathrm{NP}$, $\Pi_1^p = \mathrm{coNP}$.

(iii) $\Sigma_{k+1}^p = \mathrm{NP}^{\Pi_k^p} = \mathrm{NP}^{\Delta_{k+1}^p}$.

(iv) $\mathrm{P}^{\Delta_k^p} = \Delta_k^p$.

Proof. (i) $\Delta_1^p = \mathrm{P}^{\mathrm{P}} = \mathrm{P}$.

(ii) $\Sigma_1^p = \mathrm{NP}^{\mathrm{P}} = \mathrm{NP}$, $\Pi_1^p = \mathrm{co}\Sigma_1^p = \mathrm{coNP}$.

(iii) Let $B \in \Sigma_{k+1}^p$, $B = L(M^A)$ and $A \in \Sigma_k^p$. Further, let M' be the machine obtained from M by interchanging the states Y and N. Obviously, $B = L(M'^{\overline{A}})$ and therefore $B \in \mathrm{NP}^{\Pi_k^p}$. In addition, $\mathrm{NP}^{\Delta_{k+1}^p} = \mathrm{NP}^{\mathrm{P}^{\Sigma_k^p}} = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p$ holds.

(iv) $k = 0$: $\mathrm{P}^{\Delta_0^p} = \mathrm{P}^{\mathrm{P}} = \mathrm{P} = \Delta_0^p$.
$k > 0$: $\mathrm{P}^{\Delta_k^p} = \mathrm{P}^{\mathrm{P}^{\Sigma_{k-1}^p}} = \mathrm{P}^{\Sigma_{k-1}^p} = \Delta_k^p$. Q.E.D.

Theorem 4.6. For all k, $\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Proof. For $k = 0$, the theorem states $\mathrm{P} \subseteq \mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$. This is obviously true.

For $k > 0$:

- $\Sigma_k^p \subseteq \mathrm{P}^{\Sigma_k^p}$ and therefore also $\Pi_k^p \subseteq \mathrm{P}^{\Sigma_k^p}$ because $\mathrm{P}^{\Sigma_k^p}$ is closed under complement. Hence, $\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p$.
- $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p} = \mathrm{coP}^{\Sigma_k^p} \subseteq \mathrm{coNP}^{\Sigma_k^p} = \Pi_{k+1}^p$.
  $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p} \subseteq \mathrm{NP}^{\Sigma_n^p} = \Sigma_{k+1}^p$. Therefore, $\Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Q.E.D.

Theorem 4.7. If there is a k such that $\Sigma_{k+1}^p = \Sigma_k^p$, then $\Sigma_{k+i}^p = \Pi_{k+i}^p = \Sigma_k^p$ for all $i > 0$.

Proof. For $i = 1$, $\Sigma_{k+i}^p = \Sigma_{k+1}^p = \Sigma_k^p$ by assumption. By induction hypothesis, assume $\Sigma_{k+i}^p = \Sigma_k^p$. Then,

$$\Sigma_{k+i+1}^p = \mathrm{NP}^{\Sigma_{k+i}^p} = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p = \Sigma_k^p ,$$

and therefore also

$$\Pi_{k+i}^p \subseteq \Sigma_{k+i+1}^p = \Sigma_k^p \quad \text{for all } i.$$

In particular, $\Pi_k^p \subseteq \Sigma_k^p$ holds. It remains to show that $\Sigma_k^p \subseteq \Pi_k^p$. If $B \in \Sigma_k^p$, then $\overline{B} \in \Pi_k^p \subseteq \Sigma_k^p$ and, hence, $B \in \Pi_k^p$. Q.E.D.

Corollary 4.8. If there is a $k > 0$ with $\Sigma_k^p \neq \mathrm{P}$, then $\mathrm{P} \neq \mathrm{NP}$.

Definition 4.9. $\mathrm{PH} := \bigcup_{k \in \mathbb{N}} \Sigma_k^p$ is called the *polynomial hierarchy*.

In case $\Sigma_{k+i}^p = \Sigma_k^p$, we say that the polynomial hierarchy collapses at level k.

Theorem 4.10. $\mathrm{PH} \subseteq \textsc{Pspace}$.

Proof. By induction over k we show that $\Sigma_k^p \subseteq \textsc{Pspace}$ for all $k \in \mathbb{N}$:

$$\Sigma_0^p = \mathrm{P} \subseteq \textsc{Pspace}$$
$$\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p} \subseteq \mathrm{NP}^{\textsc{Pspace}} \subseteq \textsc{Pspace}^{\textsc{Pspace}} = \textsc{Pspace}.$$

Here, $\text{PSPACE}^{\text{PSPACE}}$ is the class of languages that can be decided by a deterministic polynomially-space bounded Oracle Turing machine with some oracle in PSPACE. When we speak about space complexity, we also count the space used on the oracle tape. Q.E.D.

If PH = PSPACE, the polynomial hierarchy collapses:

Theorem 4.11. If PH = PSPACE, there is some k with $\text{PH} = \Sigma_k^p$.

Proof. If PH = PSPACE, then QBF $\in$ PH holds. Consequently, there is some k such that QBF $\in \Sigma_k^p$. For each $A \in$ PSPACE, we have $A \leq_m^p$ QBF, i.e., $A \in \Sigma_k^p$. Thus, we obtain $\text{PH} = \Sigma_k^p$. Q.E.D.

It is assumed that $\Sigma_k^p \subsetneq \Sigma_{k+1}^p$ for all k, the polynomial hierarchy is strict and therefore, PH $\subsetneq$ PSPACE.

4.2.1 Additions

There are two natural complete problems for Σ_k^p and Π_k^p:

$$\Sigma_k\text{-QBF} = \{\psi = (\exists \overline{X_1})(\forall \overline{X_2}) \dots (Q_k \overline{X_k})\varphi \ : \ \varphi \text{ quantifier free}, \ \psi \text{ true}\}$$

Here, Q_k is the universal quantifier if k is even and the existential quantifier otherwise. Π_k-QBF is defined analogously but the formulae begin with universal quantifiers. The problem Σ_k-QBF is Σ_k^p-complete and, analogously, Π_k-QBF is Π_k^p-complete. This generalises the NP-completeness of SAT.

Recall the definition of NP. A problem $A \in$ NP if, and only if, there is some $B \in$ P and some polynomial $p(n)$ such that $A = \{x \ : \ \exists^p y(x\#y \in B)\}$ where $\exists^p y$ is an abbreviation for $\exists y \ : \ |y| \leq p(|x|)$. We can generalise this definition to obtain a characterisation of Σ_k^p and Π_k^p as follows:

- $A \in \Sigma_k^p$ if, and only if, there is some $B \in$ P and some polynomial $p(n)$ such that

$$A = \{x \ : \ (\exists^p y_1)(\forall^p y_2)(\exists^p y_3) \dots (Q_k^p y_k) x\#y_1\#y_2\# \dots \#y_k \in B\}.$$

Here, Q_k is the existential quantifier if k is odd and the universal quantifier otherwise.

- Π_k^p can be characterised analogously, using formulae that begin with universal quantifiers.

4.3 Relativisations

To approach the P = NP question, it is interesting to see whether there are oracles A, B such that

- $\mathrm{P}^A = \mathrm{NP}^A$
- $\mathrm{P}^B \neq \mathrm{NP}^B$.

The first question is easy to answer since $\mathrm{P}^{\mathrm{QBF}} = \mathrm{NP}^{\mathrm{QBF}} = \textsc{Pspace}$. The answer to the second question is not as straightforward.

Theorem 4.12. There is an oracle B such that $\mathrm{P}^B \neq \mathrm{NP}^B$.

Proof. Just as Turing machines, polynomially time-bounded oracle Turing machines can also be enumerated recursively (exercise). We choose one such recursive enumeration $\{M_i : i \in \mathbb{N}\}$ of deterministic polynomial oracle Turing machines such that the following holds for all oracles A:

(1) $\mathrm{P}^A = \{L(M_i^A) : i \in \mathbb{N}\}$.
(2) There is a sequence $\{p_i(n) : i \in \mathbb{N}\}$ of polynomials with:
  (i) M_i is p_i-time bounded,
  (ii) $p_i(n) \leq p_{i+1}(n)$ for all i, n.

Any given sequence $\{q_i(n) : i \in \mathbb{N}\}$ of time bounds for $\{M_i : i \in \mathbb{N}\}$ can be modified to such a sequence $p_{i+1}(n)$ by setting:

- $p_0(n) := q_0(n)$,
- $p_{i+1}(n) := \max(p_i(n), q_{i+1}(n))$.

For every $C \subseteq \{0,1\}^*$, let $S(C) = \{0^n : \text{there is an } x \in C \text{ with } |x| = n\}$.

We obviously have:

Lemma 4.13. $S(B) \in \mathrm{NP}^B$ for all B.

The goal now is to find some B such that $S(B) \notin \mathrm{P}^B$. This B is constructed as follows: At the beginning, initialise $B_0 := \varnothing$ and $k_0 := 0$. For $n > 0$, construct B_n, k_n as follows:

- Set k_n so it is the smallest integer with $2^{k_n} > p_n(k_n)$ and $k_n > p_{n-1}(k_{n-1})$.
- If $0^{k_n} \in L(M_n^{B_{n-1}})$, then set $B_n := B_{n-1}$. Otherwise, let $w(n)$ be the lexicographically first word in $\{0,1\}^{k(n)}$ for which the oracle is not queried during the computation of $M_n^{B_{n-1}}$ on input 0^{k_n}. Such a word exists since $p_n(k_n) < 2^{k_n}$. Set $B_n = B_{n-1} \cup \{w(n)\}$.

Set $B := \bigcup_{n \in \mathbb{N}} B_n$.

Lemma 4.14. $0^{k_n} \in L(M_n^B) \iff 0^{k_n} \in L(M_n^{B_{n-1}})$ for all n.

The oracle is never queried for $w \in B \setminus B_{n-1}$ during the computation of M_n^B on 0^{k_n}:

- for $w = w(n)$ by construction and
- for $w = w(m)$ with $m > n$ because $|w(m)| = k(m) > p_n(k_n)$.

Lemma 4.15. $S(B) \notin \mathrm{P}^B$.

Otherwise, there would be some $n \in \mathbb{N}$ with $S(B) = L(M_n^B)$. However, this cannot be the case since, by definition, $0^{k_n} \in S(B)$ if, and only if, there is some w such that $|w| = k_n$ and $w \in B$. By construction, this is the case if, and only if, $0^{k_n} \notin L(M_n^{B_{n-1}})$. This follows from the fact that a word of length k_n is added to B if, and only if, $0^{k_n} \notin L(M_n^{B_{n-1}})$. By Lemma 4.14, $0^{k_n} \notin L(M_n^{B_{n-1}})$ is equivalent to $0^{k_n} \notin L(M_n^B)$, and therefore, $S(B) \neq L(M_n^B)$. Q.E.D.

The problem whether $\mathcal{C}_1 = \mathcal{C}_2$ remains open for many pairs of complexity classes $\mathcal{C}_1$ and $\mathcal{C}_2$. In most cases, there are oracles A and B such that:

$$\mathcal{C}_1^A = \mathcal{C}_2^A \qquad \text{and} \qquad \mathcal{C}_1^B \neq \mathcal{C}_2^B.$$

It has further been shown that for almost all oracles D: $\mathcal{C}_1^D \neq \mathcal{C}_2^D$. Contradictory relativisations of this kind show that, with respect to the

problem $\mathcal{C}_1 \neq \mathcal{C}_2$, proof techniques that 'relativise' (i.e., techniques that are independent of oracles) fail.

5 Alternating Complexity Classes

Alternating algorithms are a generalization of non-deterministic algorithms based on two-player games. Indeed, one can view non-deterministic algorithms as the restriction of alternating algorithms to solitaire (i.e., one-player) games. Since complexity classes are mostly defined in terms of Turing machines, we focus on the model of alternating Turing machines. But note that alternating algorithms can be defined in terms of other computational models, also.

Definition 5.1. An *alternating Turing machine* is a non-deterministic Turing machine whose state set Q is divided into four classes $Q_\exists$, $Q_\forall$, Q_{acc}, and Q_{rej}. This means that there are existential, universal, accepting and rejecting states. States in $Q_{acc} \cup Q_{rej}$ are final states. A configuration of M is called existential, universal, accepting, or rejecting according to its state.

The computation graph $G_{M,x}$ of an alternating Turing machine M for an input x is defined in the same way as for a non-deterministic Turing machine. Nodes are configurations (instantaneous descriptions) of M, there is a distinguished starting node $C_0(x)$ which is the input configuration of M for input x, and there is an edge from configuration C to configuration C' if, and only if, C' is a successor configuration of C. Recall that for *non-deterministic* Turing machines, the acceptance condition is given by the reachability problem: M accepts x if, and only if, in the graph $G_{M,x}$ some accepting configuration C_a is reachable from $C_0(x)$. For *alternating* Turing machines, acceptance is defined by the GAME problem (see Sect. 3.3): the players here are called $\exists$ and $\forall$, where $\exists$ moves from existential configurations and $\forall$ from universal ones. Further, $\exists$ wins at accepting configurations and loses at rejecting ones. By definition, M accepts x if, and only if, Player $\exists$ has a winning strategy from $C_0(x)$ for the game on $G_{M,x}$.

When considering the computation tree $\mathfrak{T}_{M,x}$, which corresponds to the unraveling of the configuration graph from $C_0(x)$, we call a subtree T_C accepting if $\exists$ has a winning strategy from C.

5.1 Complexity Classes

Time and space complexity are defined as for nondeterministic Turing machines. For a function $F : \mathbb{N} \to \mathbb{R}$, we say that an alternating Turing machine M is F-time-bounded if for all inputs x, all computation paths from $C_0(x)$ terminate after at most $F(|x|)$ steps. Similarly, M is F-space-bounded if no configuration of M that is reachable from $C_0(x)$ uses more than $F(|x|)$ cells of work space. The complexity classes $\textsc{Atime}(F)$ and $\textsc{Aspace}(F)$ contain all problems that are decidable by, respectively, F-time bounded and F-space bounded alternating Turing machines.

The following classes are of particular interest:

- $\textsc{Alogspace} = \textsc{Aspace}(O(\log n))$,
- $\textsc{Aptime} = \bigcup_{d\in\mathbb{N}} \textsc{Atime}(n^d)$,
- $\textsc{Apspace} = \bigcup_{d\in\mathbb{N}} \textsc{Aspace}(n^d)$.

Example 5.2. $\textsc{qbf} \in \textsc{Atime}(O(n))$. We assume that, without loss of generality, negations appear only in front of variables. An alternating version of $\text{Eval}(\psi, \mathfrak{I})$ is the following:

Algorithm 5.1. Alternating $\text{Eval}(\psi, \mathfrak{I})$

Input: $(\psi, \mathfrak{I})$ where $\psi \in \textsc{qbf}$ und $\mathfrak{I}$: $\text{free}(\psi) \to \{0,1\}$
if $\psi = Y$ **then**
  if $\mathfrak{I}(Y) = 1$ **then** accept **else** reject
endif
if $\psi = \varphi_1 \vee \varphi_2$ **then** "$\exists$" guesses $i \in \{1,2\}$; **return** $\text{Eval}(\varphi_i, \mathfrak{I})$
if $\psi = \varphi_1 \wedge \varphi_2$ **then** "$\forall$" chooses $i \in \{1,2\}$; **return** $\text{Eval}(\varphi_i, \mathfrak{I})$
if $\psi = \exists X\varphi$ **then** "$\exists$" guesses $j \in \{0,1\}$; **return** $\text{Eval}(\varphi, \mathfrak{I}[X = j])$
if $\psi = \forall X\varphi$ **then** "$\forall$" chooses $j \in \{0,1\}$; **return** $\text{Eval}(\varphi, \mathfrak{I}[X = j])$

5.2 Alternating Versus Deterministic Complexity

There is a general slogan that parallel time complexity coincides with sequential space complexity.

Theorem 5.3. Let $S(n)$ be space-constructible with $S(n) \geq n$. Then,

$$\textsc{Nspace}(S) \subseteq \textsc{Atime}(S^2).$$

Proof. We use the same trick as in the proof of Savitch's Theorem: Let L be decided by a nondeterministic Turing machine M with space bounded by $S(n)$ and in time $2^{c \cdot S(n)}$. Let Conf$[S(n)]$ be the set of configurations of M with space $\leq S(n)$. The alternating algorithm Reach(C_1, C_2, t) (Algorithm 5.2) decides whether the configuration $C_2 \in$ Conf$[S(n)]$ can be reached from configuration $C_1 \in$ Conf$[S(n)]$ in at most 2^t steps. The algorithm is correct because C_2 is reachable from C_1 in at most 2^t steps if there is some C such that Reach$(C_1, C, t-1)$ and Reach$(C, C_2, t-1)$ accept.

Let $f(t) = \max_{C_1, C_2 \in \text{Conf}[S(n)]} \text{time}_{\text{Reach}}(C_1, C_2, t)$. Furthermore, $f(0) = O(S(n))$ and for all $t > 0$, $f(t) = O(S(n)) + f(t-1)$. Hence,

$$f(t) = (t+1) \cdot O(S(n)).$$

L can then be decided as follows: At first, for an input x, the input configuration C_0 of M on x is constructed. Then, some accepting final configuration C_a of M is guessed. We will accept if Reach$(C_0, C_a, S(n))$

Algorithm 5.2. Reach(C_1, C_2, t)

Input: C_1, C_2, t
if $t = 0$ **then**
  if $C_1 = C_2$ *or* $C_2 \in \mathit{Next}(C_1)$ **then** accept **else** reject
else /* $t > 0$ */
  existentially guess $C \in$ Conf$[S(n)]$
  universally choose $(D_1, D_2) = (C_1, C)$ and $(D_1, D_2) = (C, C_2)$
  Reach$(D_1, D_2, t-1)$
endif

accepts. This algorithm needs

$$(c \cdot S(n) + 1)O(S(n)) = O(S^2(n))$$

steps. By the linear Speed-Up Theorem, which also applies to alternating Turing machines, $L \in \text{ATIME}(S^2)$. Q.E.D.

Theorem 5.4. Let T be space-constructible and $T(n) \geq n$. Then, $\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$.

Proof. Let $L \in \text{ATIME}(T)$ and M be some alternating Turing machine accepting L in time bounded by $T(n)$. Then, there is some r so that for all configurations C of M: $|\text{Next}(C)| \leq r$. Algorithm 5.3, $\mathcal{A}_T$, computes whether or not the subtree T_C is accepting (output 1) or rejecting (output 0) for every configuration C in $\mathfrak{T}_{M,x}$.

Obviously, this algorithm is working correctly. $\mathcal{A}_T(C_0(x))$ decides whether M accepts x and, hence, is a deterministic decision procedure for L.

Algorithm 5.3. $\mathcal{A}_T$, deterministic evaluation of T_C

Input: C
if C *accepting* **then output** 1
if C *rejecting* **then output** 0
if C *existential* **then**
    for $i = 1, \ldots, r$ **do**
        compute i-th successor configuration C_i of C
        if $F(C_i) = 1$ **then output** 1
    endfor
    output 0
endif
if C *universal* **then**
    for $i = 1, \ldots, r$ **do**
        compute i-th successor configuration C_i of C
        if $F(C_i) = 0$ **then output** 0
    endfor
    output 1
endif

How much space does this algorithm need? Let C be some node of height t in $\mathfrak{T}_{M,x}$, i.e., all computations of M rooted at C need at most t steps. Then:

$$\text{space}_{\mathcal{A}_T}(C) = \begin{cases} 0 & \text{if } t = 0 \\ \max_{C_i \in \text{Next}(C)} (|C_i| + \text{space}_{\mathcal{A}_T}(C_i)) & \text{if } t > 0\,. \end{cases}$$

Since $C_i \in \text{Next}(C)$ is of height $t-1$, we obtain $\text{space}_{\mathcal{A}_T}(C) \leq t \cdot T(n)$ and therefore $\text{space}_{\mathcal{A}_T}(C_o) \leq T^2(n)$. Q.E.D.

In particular, we obtain

Theorem 5.5 (Parallel time complexity = sequential space complexity)**.**

- APTIME = PSPACE.
- AEXPTIME = EXPSPACE.

Proof.

- $\text{ATIME}(n^d) \subseteq \text{DSPACE}(n^{2d}) \subseteq \text{PSPACE}$,
  $\text{DSPACE}(n^d) \subseteq \text{NSPACE}(n^d) \subseteq \text{ATIME}(n^{2d}) \subseteq \text{APTIME}$.
- $\text{ATIME}(2^{n^d}) \subseteq \text{DSPACE}(2^{2n^d}) \subseteq \text{EXPSPACE}$,
  $\text{DSPACE}(2^{n^d}) \subseteq \text{ATIME}(2^{2n^d}) \subseteq \text{AEXPTIME}$. Q.E.D.

On the other hand, alternating space complexity corresponds to exponential deterministic time complexity.

Theorem 5.6. For any space-constructible function $S(n) \geq \log n$, we have that $\text{ASPACE}(S) = \text{DTIME}(2^{O(S)})$.

Proof. The proof is closely associated with the GAME problem. For any S-space-bounded alternating Turing machine M, one can, given an input x, construct the computation graph $G_{M,x}$ in time $2^{O(S(|x|)}$ and then solve the GAME problem in order to decide the acceptance of x by M.

For the converse, we shall show that for any $T(n) \geq n$ and any constant c, $\text{DTIME}(T) \subseteq \text{ASPACE}(c \cdot \log T)$.

Let $L \in \text{DTIME}(T)$. Then there is a deterministic one-tape Turing machine M that decides L in time T^2. Let $\Gamma = \Sigma \cup (Q \times \Sigma) \cup \{*\}$ and

$t = G^2(n)$. Every configuration $C = (q, i, w)$ (in a computation on some input of length n) can be described by a word

$$\underline{c} = *w_0 \dots w_{i-1}(qw_i)w_{i+1} \dots w_t * \in \Gamma^{t+2}.$$

The ith symbol of the successor configuration depends only on the symbols at positions $i-1$, i, and $i+1$. Hence, there is a function $f_M : \Gamma^3 \to \Gamma$ such that, whenever symbols a_{-1}, a_0, and a_1 are at positions $i-1$, i and $i+1$ of some configuration $\underline{c}$, the symbol $f_M(a_{-1}, a_0, a_1)$ will be at position i of the successor configuration $\underline{c}'$.

The alternating algorithm $\mathcal{A}$ (Algorithm 5.4) decides L using space $O(\log T(n))$. If M accepts the input x, then Player $\exists$ has the following winning strategy for the game on $C_{\mathcal{A},x}$: the value chosen for s is the time at which M accepts x, and $(q^+a), i$ are chosen so that the configuration of M at time s is of the form $*w_0 \dots w_{i-1}(q^+a)w_{i+1} \dots w_t*$. At the jth iteration of the loop (that is, at configuration $s-j$), the symbols at positions $i-1, i, i+1$ of the configuration of M at time $s-j$ are chosen for a_{-1}, a_0, a_1.

Conversely, if M does not accept the input x, the ith symbol of the configuration at time s is not (q^+a). The following holds for all j: if, in the jth iteration of the loop, Player $\exists$ chooses a_{-1}, a_0, a_1, then

Algorithm 5.4. Alternating simulation of a determinisitc computation

existentially guess $s \leq T^2(n) = t$
existentially guess $i \in \{0, \dots, s\}$
existentially guess $(q^+a) \in Q^+_{acc} \times \Sigma$
$b := (q^+a)$
for $j = 1, \dots, s$ **do**
    existentially guess $(a_{-1}, a_0, a_1) \in \Gamma^3$
    if $f_M(a_{-1}, a_0, a_1) \neq b$ **then** reject
    universally choose $k \in \{-1, 0, 1\}$
    $b := a_k$
    $i := i + k$
endfor
if *the i-th symbol of the input configuration of M on x equals b* **then** accept
else reject

either $f(a_{-1}, a_0, a_1) \neq b$, in which case Player $\exists$ loses immediately, or there is at least one $k \in \{-1, 0, 1\}$ such that the $(i+k)$th symbol of the configuration at time $s - j$ differs from a_k. Player $\forall$ then chooses exactly this k. At the end, a_k will then be different from the ith symbol of the input configuration, so Player $\forall$ wins.

Hence $\mathcal{A}$ accepts x if, and only if, M does so. Q.E.D.

In particular, it follows that

- ALOGSPACE = PTIME;
- APSPACE = EXPTIME.

The relationship between the major deterministic and alternating complexity classes is summarised in Fig. 5.1.

$$\begin{array}{ccccccccc} \text{LOGSPACE} & \subseteq & \text{PTIME} & \subseteq & \text{PSPACE} & \subseteq & \text{EXPTIME} & \subseteq & \text{EXPSPACE} \\ & & \| & & \| & & \| & & \| \\ & & \text{ALOGSPACE} & \subseteq & \text{APTIME} & \subseteq & \text{APSPACE} & \subseteq & \text{AEXPTIME} \end{array}$$

Figure 5.1. Relationship between deterministic and alternating complexity classes

5.3 Alternating Logarithmic Time

For time bounds $T(n) < n$, the standard model of alternating Turing machines needs to be modified a little by an indirect access mechanism. The machine writes down, in binary, an address i on an separate index tape to access the ith symbol of the input. Using this model, it makes sense to define, for instance, the complexity class $\text{ALOGTIME} = \text{ATIME}(O(\log n))$.

Important examples of problems in ALOGTIME are

- the model-checking problem for propositional logic;
- the data complexity of first-order logic.

The results mentioned above relating alternating time and sequential space hold also for logarithmic time and space bounds. Note, however, that these do not imply that ALOGTIME = LOGSPACE, owing to

the quadratic overheads. It is known that ALogtime $\subseteq$ Logspace, but the converse inclusion is an open problem.

Exercise 5.1. Construct an ALogtime algorithm for the set of palindromes (i.e., words that are same when read from right to left and from left to right).

6 Complexity Theory for Probabilistic Algorithms

Probabilistic algorithms are algorithms that can, at certain points during their computation, choose one possibility for the next operation *at random* from a number of different possibilities. They can thus be seen as a modification of nondeterministic algorithms. The computation result of such an algorithm therefore is not a definite answer but a random variable: it depends on the decisions made "at random" during the computation. Please note that this has nothing to do with an assumption on the distribution of possible inputs. The probability does not concern the inputs but rather the decisions during the computation.

Probabilistic algorithms play an important role in many different areas. They are often simpler and more efficient than the best known deterministic algorithms for the same problem. Even more, some important areas such as algorithmic number theory or cryptology are inconceivable without probabilistic algorithms. We will look at two examples.

6.1 Examples of probabilistic algorithms

6.1.1 Perfect matching and symbolic determinants

We first recall the definition of the marriage problem. Given is a bipartite graph $G = (U, V, E)$ with two disjoint sets of nodes $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$ of the same size and a set of edges $E \subseteq U \times V$. The problem is to determine whether G permits a *perfect matching*, i.e., a subset $M \subseteq E$ such that for all $u \in U$ there is a $v \in V$ and for all $v \in V$ there is a $u \in U$ such that $(u, v) \in M$. We can rephrase the problem

like this: Is there a permutation $\pi \in S_n$ so that $(u_i, v_{\pi(i)}) \in E$ for all $i \in \{1, \ldots, n\}$?

The marriage problem can further be described as a problem over matrices and determinants. The graph $G = (U, V, E)$ is then characterised by a matrix A^G whose items are variables x_{ij} or 0.

$$A^G := (z_{ij})_{1 \le i,j \le n} \text{ with } z_{ij} := \begin{cases} x_{ij} & \text{if } (u_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

The determinant of A^G is $\det A^G := \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^n z_{i\pi(i)}$ where $\operatorname{sgn}(\pi) = 1$ if π is the product of an even number of transpositions and $\operatorname{sgn}(\pi) = -1$ otherwise. Obviously, $\det A^G$ is a polynomial in $\mathbb{Z}[x_{11}, \ldots, x_{nn}]$ (i.e., a polynomial with coefficients in $\mathbb{Z}$) of total degree n that is linear in every variable x_{ij}.

A permutation $\pi \in S_n$ defines a perfect matching if and only if $\prod_{i=1}^n z_{ij} \neq 0$. Since all of these products are pairwise different, we obtain

$$G \text{ allows a perfect matching} \iff \det A^G \neq 0.$$

Hence, if we were able to compute symbolic determinants (i.e., determinants of matrices that can contain variables) efficiently, we could use this to solve the marriage problem.

Determinants using Gauss elimination. We know from linear algebra how to compute determinants from numerical matrices: the given matrix is transformed (e.g., by interchanging lines or by adding linear combinations of lines to other lines) into a triangular matrix that has the same determinant. The products of the diagonal elements are then calculated to obtain the determinant. This requires $O(n^3)$ arithmetical operations. Further, the entries of the transformed matrices remain polynomially-bounded since they are subdeterminants of the given matrix.

Unfortunately, the application of this procedure to symbolic matrices is problematic. The entries of the transformed matrices are rational

functions in the entries of the original matrix and these functions generally have exponentially many terms. Even the problem whether a fixed mononom, e.g., $x_{11}x_{23}x_{31}$, appears in the determinant of A^G is NP-hard. Hence, Gauss elimination does not seem to be useful to calculate symbolic determinants.

However, we do not need to compute the determinant of A^G. It suffices to know whether it is 0 or not. The idea for the probabilistic algorithm solving the perfect matching problem is to substitute a tuple $\bar{a} = (a_{11}, \ldots, a_{nn})$ of random numbers into the matrix A^G and then to calculate the determinant of the numerical matrix $A^G(\bar{a})$ using Gauss elimination.

If we obtain that $\det A^G(\bar{a}) \neq 0$ then the symbolic determinant $\det A^G$ is obviously not 0. The inverse does not hold: It might be the case that we incidentally find a root $\bar{a}$ of $\det A^G$ and, hence, obtain $\det A^G \neq 0$.

The following lemma allows us to control the probability to obtain the roots of a non-identically disappearing polynomial $\det A^G$ by finding a suitable set to choose $\bar{a}$ from.

Lemma 6.1. Let $p(x_1, \ldots, x_n)$ be a polynomial such that $p \neq 0$ and every x_i is at most of degree d in p. Then, for every $m \in \mathbb{N}$,

$$|\{(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n : p(a_1, \ldots, a_n) = 0\}| \leq ndm^{n-1}.$$

Proof. We will use induction over n. For $n = 1$, the induction hypothesis is a known fact: no polynomial $p \neq 0$ with one variable of degree d has more than d roots. Further, consider $n > 1$. We write $p(x_1, \ldots, x_n)$ as a polynomial in x_n with coefficients from $\mathbb{Z}[x_1, \ldots, x_{n-1}]$:

$$p(x_1, \ldots, x_n) = p_0(x_1, \ldots, x_{n-1}) + p_1(x_1, \ldots, x_{n-1})x_n + \cdots + p_d(x_1, \ldots, x_{n-1})x_n^d.$$

Let now $p(a_1, \ldots, a_n) = 0$ for $(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n$. We consider two cases:

(a) $p_d(a_1, \ldots, a_{n-1}) = 0$. By induction hypothesis, this is the case for at most $(n-1)dm^{n-2}$ tuples $(a_1, \ldots, a_{n-1}) \in \{0, \ldots, m-1\}^{n-1}$. Thus,

there are at most $(n-1)dm^{n-1}$ roots $(a_1,\ldots,a_n) \in \{0,\ldots,m-1\}^n$ of p with $p_d(a_1,\ldots,a_{n-1}) = 0$.

(b) $p_d(a_1,\ldots,a_{n-1}) \neq 0$. Then, $p(a_1,\ldots,a_{n-1},x_n)$ is a polynomial of degree d in variables x_n, for which there are at most d roots a_n. In addition to the roots in case (a), there are, hence, at most dm^{n-1} new roots.

Hence, we have at most ndm^{n-1} roots $(a_1,\ldots,a_n) \in \{0,\ldots,m-1\}^n$. Q.E.D.

Consequently, we obtain a probabilistic algorithm for the perfect matching problem.

Input: a matrix A^G for a bipartite graph $G = (U,V,E)$, $|U| = |V| = n$
a security parameter $k \in \mathbb{N}$
Set $m := 2n^2$
for $i = 1,\ldots,k$ **do**
Choose at random numbers $a_{11},\ldots,a_{nn} \in \{0,\ldots,m-1\}$
Compute $\det A^G(\bar{a})$ using Gauss elimination
if $\det A^G(\bar{a}) \neq 0$ **then output** 'There is a perfect matching'
endfor
output 'There is probably no perfect matching'

Since the computation of numerical determinants can be done in polynomial-time using Gauss elimination, this is also a polynomial-time algorithm. If the algorithm finds a tuple $\bar{a}$ such that $\det A^G \neq 0$, it will return 'There is a perfect matching' and this is correct. If it does not find such a $\bar{a}$ after k iterations, it will return 'There is probably no perfect matching'. This, however, is not always correct. The error probability, i.e., the probability that the algorithm does not find a non-root for a non-disappearing polynomial $\det A^G$, can be estimated using the above lemma.

Since $\det A^G$ is linear in each of the n^2 variables, the ratio of tuples $\bar{a} \in \{0,\ldots,m-1\}^{n^2}$ that are roots of $\det A^G$ is at most

$$\frac{n^2 dm^{n^2-1}}{m^{n^2}} = \frac{n^2 d}{m} = \frac{n^2}{2n^2} = \frac{1}{2}.$$

The probability to find only such tuples in k iterations is at most 2^{-k}. Please note this is not a probability statement with respect to bipartite graphs or symbolic determinants. It is indeed a statement on the error probability of a probabilistic algorithm with respect to its random decisions and is valid for all bipartite graphs.

6.1.2 A probabilistic prime number test

Two central problems of algorithmic number theory are the existence of polynomial algorithms for

(1) Primality testing: given an integer $n \in \mathbb{N}$, determine whether it is prime;
(2) Factoring: given an integer $n \in \mathbb{N}$, calculate its factorisation (its prime factors).

Primality testing has a long history going back to ancient Greece. The first systematic approach, the *Sieve of Eratosthenes*, where multiples of primes are successively removed from a list of numbers leaving only the primes, dates back to around 240 BC. While being based on multiplication only, this approach yields an algorithm that is still exponential in the size of the input like the naïve approach.

Obviously, PRIMES $\in$ coNP since each non-trivial factor is a polynomial witness for compositeness. In 1974, Pratt could prove membership in NP with some more effort.

A year later, Miller presented a deterministic polynomial-time algorithm based on Fermat's Little Theorem, but its correctness depends on the assumption of the Extended Riemann Hypothesis. In 1980, Rabin modified this test and obtained an unconditional but randomised polynomial-time algorithm, thus placing the problem in coRP. Later, in 1987, Adleman and Huang proved the quite involved result that PRIMES $\in$ RP, and hence in ZPP.

Only recently, Agrawal, Kayal and Saxena presented a deterministic polynomial-time algorithm based on a generalisation of Fermat's Little Theorem. The first version of their algorithm had a running-time in $O(n^{12})$, which could be improved to $O(n^{7.5})$, and lately to $O(n^6)$. Depending on some number-theoretic hypotheses, the running time

might be further improved to $O(n^3)$. For details see [Agrawal, Kayal, Saxena. PRIMES is in P. Annals of Mathematics 160 (2004)].

However, the currently long running-time renders this algorithm practically unusable since it is outperformed by the simple and efficient probabilistic methods which are able to determine with an almost arbitrarily high probability whether a given number is prime.

Unfortunately, neither of these methods can be used to efficiently obtain a factorisation for composite numbers. In fact, it is widely assumed that the factorisation of integers is difficult in practice, and many modern public-key cryptology systems are based on this assumption.

In the following we will present the randomised primality test due to Rabin and Miller which is based on *Fermat's Little Theorem*.

Definition 6.2. For $n \in \mathbb{N}$, let

$$\mathbb{Z}_n^* := \{a \in \{1, \ldots, n-1\} : \gcd(a, n) = 1\}.$$

Note that $(\mathbb{Z}_n^*, \cdot \pmod n)$ is a group.

Theorem 6.3 (Fermat)**.** Let p be prime. Then, for all $a \in \mathbb{Z}_p^*$,

$$a^{p-1} \equiv 1 \pmod p .$$

Proof. Let $f(p, a)$ be the number of different non-periodic colourings of cycles of length p with a colours. Since p is prime and the period must be a divisor of p for every *periodic* colouring, only periods of length 1 are possible, that is, only monochrome colourings. The number of colourings of p nodes with a colours is a^p, the number of monochrome colourings is a and, hence, $f(p, a) = (a^p - a)/p = a(a^{p-1} - 1)/p$. We obtain that p is a divisor of $a^{p-1} - 1$ and therefore, $a^{p-1} \equiv p \pmod p$. Q.E.D.

One might hope that also the inverse holds, i.e., for every *composed* number n, there is an $a \in \mathbb{Z}_n^*$ such that $a^{n-1} \not\equiv 1 \pmod n$. If one could show furthermore that there are "many" $a \in \mathbb{Z}_n^*$ with this property, a prime number test could work as follows: Given some n, it would choose an $a \in \mathbb{Z}_n^*$ at random. Then, it would check whether $a^{n-1} \equiv 1$

$(\text{mod } n)$. For this approach to work, we need to be able to verify whether $a^{n-1} \not\equiv 1 \pmod{n}$ in polynomial time (with respect to the length of the input, i.e., $\log n$). This can be done by repeating the square operation modulo n: For $k = \lfloor \log n \rfloor$, compute the numbers $b_0, \ldots, b_k$ with $b_0 := a$, $b_{i+1} := (b_i)^2 \pmod{n}$, i.e., $b_i = a^{2^i} \pmod{n}$. Let $n-1 = \sum_{i=1} u_i 2^i$ be the binary representation of $n-1$, with $u_i \in \{0,1\}$. Then,

$$a^{n-1} = a^{\sum_i u_i 2^i} = \prod_i a^{u_i 2^i} \equiv \prod_{u_i=1} b_i \pmod{n}.$$

Unfortunately, the Fermat test in this simple form fails. This is because the inversion of Fermat's Little Theorem is incorrect. There are (even infinitely many) composites $n \in \mathbb{N}$ such that $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}_n^*$. These numbers are called *Carmichael numbers*. The first Carmichael numbers are 561 and 1729.

The idea works, however, for every non-Carmichael number. For $n \in \mathbb{N}$, let

$$F_n := \{a \in \mathbb{Z}_n^* : a^{n-1} \equiv 1 \pmod{n}\}.$$

Lemma 6.4. If n is composite and not a Carmichael number, then $|F_n| \leq |Z_n^*|/2$.

Proof. It is easy to see that $(F_n, \cdot \pmod{n})$ is a subgroup of $(\mathbb{Z}_n^*, \cdot \pmod{n})$. Since n is neither prime nor a Carmichael number, $F_n \subsetneq Z_n^*$. The order of a subgroup is always a divisor of the order of the group, i.e., $|\mathbb{Z}_n^*| = q|F_n|$ for some $q \geq 2$. Q.E.D.

Hence, the fact that our original idea for a prime number test does not work is simply due to the Carmichael numbers. It is, however, possible to refine the Fermat test and treat Carmichael numbers properly. There are two variants of such probabilistic primality tests, the Solovay-Strassen test and the Rabin-Miller test, which will be described in the following. It is based on the following observation.

Lemma 6.5. Let p be prime. Then, for all $a \in \mathbb{Z}_p^*$ if $a^2 \equiv 1 \pmod{p}$, then $a \equiv \pm 1 \pmod{p}$.

Proof. If p is prime, then $(\mathbb{Z}_p^*, + \pmod n, \cdot \pmod n)$ is a field and fields have only the trivial roots 1 and -1. Q.E.D.

Theorem 6.6.

(i) The Rabin-Miller primality test (Algorithm 6.1) can be performed in polynomial-time (with respect to $\log n$).

(ii) If n is prime, the test always returns "n is probably prime".

(iii) If n is composite, the test returns "n is composite" with a probability of $\geq 1 - 2^{-k}$.

Hence, the result "n is composed" is always correct, and the answer "n is probably prime" means that n is indeed prime with a very high probability.

Proof. Proposition (i) is obviously correct. Proposition (ii) results from Theorem 6.3 and Lemma 6.5. If n is prime, then for all a used in the test:

- $a^{n-1} \equiv 1 \pmod n$.
- $b_j \not\equiv 1 \pmod n$ but $b_{j+1} = (b_j)^2 \equiv 1 \pmod n$.
  Hence, $b_j \equiv -1 \pmod n$

We obtain that the test returns "n is probably prime".

Algorithm 6.1. The Rabin-Miller primality test

Input: an odd number $n \in N$
a security parameter k
Compute t, w such that $n - 1 = 2^t w$ with w odd
for k *times* **do**
Choose $a \in \{1, \ldots, n-1\}$ at random
Compute $b_i := a^{2^i w} \pmod n$ for $i = 0, \ldots, t$
if $b_t = a^{n-1} \not\equiv 1 \pmod n$ **then output** "n is composite"
Determine $j := \max\{i : b_i \not\equiv 1 \pmod n\}$
if $b_j \not\equiv -1 \pmod n$ **then output** "n is composite"
endfor
output "n probably prime"

As for Proposition (iii), let M_n be the set of all $a \in \{1, \ldots, n-1\}$ such that the choice of a by the Rabin-Miller test with input n does *not* lead to the result "n is composed". It obviously suffices to show that $|M_n| \leq (n-1)/2$ for all composed, odd n. The probability to obtain only elements $a \in M_n$ when choosing k times some random a is smaller than 2^{-k}.

We see that $M_n \subseteq \mathbb{Z}_n^*$. If indeed $a \in M_n$ then $a^{n-1} \equiv 1 \pmod{n}$ and, hence, $a^{n-2}a + rn = 1$ for a suitable $r \in \mathbb{Z}$. If a and n had a common divisor $q > 1$, it would also be a divisor of the sum $a^{n-2}a + rn$ which is impossible since it is equal to 1. Therefore, a and n are co-prime and thus $a \in \mathbb{Z}_n^*$. Hence, it suffices to show the following.

Claim 6.7. There is a proper subgroup $U_n < Z_n^*$ which contains M_n.

From this, we obtain $|M_n| \leq |U_n| \leq |\mathbb{Z}_n^*|/2 \leq (n-1)/2$.

For composed non-Carmichael numbers n, the claim follows directly from Lemma 6.4 since $M_n \subseteq F_n$. For Carmichael numbers, we first show that these are not powers of primes, i.e., every Carmichael number n can be written as the product of two co-prime odd numbers n_1, n_2. Fix such an $n = n_1 \cdot n_2$.

For every $a \in M_n$, the sequence $b_0, \ldots, b_t$ (with $b_i = a^{2^i w} \pmod{n}$) has the form

$$* * * \cdots * \; -1\, 1\, 1 \cdots 1 \quad \text{or} \quad 1\, 1 \cdots 1.$$

Set

$$h := \max\{i : 0 \leq i \leq t, \text{there is an } a \in Z_n^* \text{ with } a^{2^i w} \equiv -1 \pmod{n}\}.$$

Such an h exists since, for example, $(-1)^{2^0 w} = -1$. Let now

$$U_n := \{a \in \mathbb{Z}_n^* : a^{2^h w} \equiv \pm 1 \pmod{n}\}.$$

Obviously, U_n is a subgroup of Z_n^* containing M_n. We now show that $U_n \subsetneq Z_n^*$ as follows: Let $b \in Z_n^*$ such that $b^{2^h w} \equiv -1 \pmod{n}$. By the Chinese Remainder Theorem, there is an $a \in \mathbb{Z}_n^*$ such that

(1) $a \equiv b \pmod{n_1}$, and
(2) $a \equiv 1 \pmod{n_2}$.

We show that $a \notin U_n$ by leading the claim $a \in U_n$ to a contradiction.

At first, let us consider $a \in U_n$ since $a^{2^h w} \equiv 1 \pmod{n}$. Then, also $a^{2^h w} \equiv 1 \pmod{n_1}$. However, because of (1) $a^{2^h w} \equiv b^{2^h w} \equiv -1 \pmod{n_1}$ holds, which is impossible since $n_1 > 2$.

The other possibility is that $a \in U_n$ since $a^{2^h w} \equiv -1 \pmod{n}$. Then, $a^{2^h w} \equiv -1 \pmod{n_2}$. However, because of (2) $a^{2^h w} \equiv 1 \pmod{n_2}$, which is impossible since $n_2 > 2$. Q.E.D.

Miller showed that, under the assumption of the *Extended Riemann Hypothesis* (ERH), this test yields a deterministic polynomial-time algorithm witnessing PRIMES $\in$ P.

Theorem 6.8 (Miller)**.** The ERH implies that there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n)$ is bounded by a polynomial in $\log n$ such that, for all odd *non-prime* numbers $n > 2$, one of the following is true:

(i) n is a prime power;
(ii) there is an $a < f(n)$ with $a \notin M_n$, i.e., the use of a in the Rabin-Miller test on input n leads to the result "n is composed".

Corollary 6.9. The ERH implies PRIMES $\in$ P.

6.2 Probabilistic complexity classes and Turing machines

For $m \in \mathbb{N}$, we consider $\{0,1\}^m$ as a *probability space* with *uniform distribution*: For every $u \in \{0,1\}^m$, the probability

$$\Pr_{y \in \{0,1\}^m}[y = u] = \frac{1}{2^m}.$$

Definition 6.10. A *probabilistic Turing machine* (PTM) is a Turing machine whose input consists of a pair $(x, y) \in \Sigma^* \times \{0,1\}^*$. Here, $x \in \Sigma^*$ denotes the actual input and $y \in \{0,1\}^*$ a random word controlling the computation of the machine.

A PTM M is called $p(n)$-time bounded if M stops after at most $p(|x|)$ steps on input (x, y). Without loss of generality, we can assume that $|y| = p(|x|)$.

Let M be $p(n)$-time bounded. If we consider M as an acceptor over $\Sigma^* \times \{0,1\}^*$, we obtain the language $L(M) \subseteq \Sigma^* \times \{0,1\}^*$. Hence, we define M as a *probabilistic acceptor* over Σ^*. For $x \in \Sigma^*$ with $|x| = n$, we set:

$$\begin{aligned} \Pr[M \text{ accepts } x] := &\quad \Pr_{y \in \{0,1\}^{p(n)}}[(x,y) \in L(M)] \\ = &\quad \frac{|\{y \in \{0,1\}^{p(n)} \ : \ (x,y) \in L(M)\}|}{2^{p(n)}}. \end{aligned}$$

Lemma 6.11. A language $A \subseteq \Sigma^*$ is in NP if and only if there is a polynomial PTM M such that $A = \{x \in \Sigma^* \ : \ \Pr[M \text{ accepts } x] > 0\}$.

Proof. Consider $A \in$ NP. Then, there is a $B \in$ P and a polynomial $p(n)$ such that $A = \{x \in \Sigma^* \ : \ \exists y \ (|y| \leq p(|x|) \ \wedge \ (x,y) \in B\}$. It is not difficult to modify B and $p(n)$ in a way that $A = \{x \in \Sigma^* \ : \ (\exists y \in \{0,1\}^{p(|x|)}) \ (x,y) \in B\}$. Let M be a polynomial, deterministic TM over $\Sigma^* \times \{0,1\}^*$ with $L(M) = B$. If we consider M as a probabilistic TM over Σ^*, we obtain:

$$\begin{aligned} A &= \{x \in \Sigma^* \ : \ \Pr_{y \in \{0,1\}^{p(n)}}[(x,y) \in L(M)] > 0\} \\ &= \{x \in \Sigma^* \ : \ \Pr[M \text{ accepts } x] > 0\}. \end{aligned}$$

Consider now $A = \{x \in \Sigma^* \ : \ \Pr[M \text{ accepts } x] > 0\}$ for a polynomial PTM M. Hence, for some suitable polynomial p, $A = \{x \in \Sigma^* \ : \ \Pr_{y \in \{0,1\}^{p(n)}}[(x,y) \in L(M)] > 0\}$. Then, $B := \{(x,y) \in \Sigma^* \times \{0,1\}^{p(|x|)} \ : \ (x,y) \in L(M)\}$ in P and therefore, $A = \{x \in \Sigma^* : \exists y \ (|y| \leq p(|x|) \ \wedge \ (x,y) \in B\}$ in NP. Q.E.D.

The probability to find a suitable witness y for an NP problem on input x simply by guessing can be very small. "Good" probabilistic algorithms are successful in guessing, i.e., they guess suitable witnesses with a high probability. We call a probabilistic algorithm for A stable, if $\Pr[M \text{ accepts } x]$ for $x \in A$ is *significantly* larger than $\Pr[M \text{ accepts } x]$ for $x \notin A$.

Definition 6.12. Consider a language $A \subseteq \Sigma^*$.

- $A \in$ PP (**p**robabilistic **p**olynomial time), if there is a polynomial PTM M such that

$$A = \{x \ : \ \Pr[M \text{ accepts } x] > \frac{1}{2}\}.$$

- $A \in$ BPP (**b**ounded error **p**robabilistic **p**olynomial time), if there is a polynomial PTM M such that

$$x \in A \implies \Pr[M \text{ accepts } x] \geq \frac{2}{3} \text{ and}$$
$$x \notin A \implies \Pr[M \text{ accepts } x] \leq \frac{1}{3}.$$

Probabilistic algorithms are subject to two kinds of *error probabilities*:

(1) *Incorrect positive:* $x \notin A$ but $\Pr[M \text{ accepts } x] > 0$.
(2) *Incorrect negative:* $x \in A$ but $\Pr[M \text{ accepts } x] < 1$, i.e., $\Pr[M \text{ does not accept } x] = 1 - \Pr[M \text{ accepts } x] > 0$.

We obtain the following picture for the complexity classes defined so far:

BPP: both error probabilities $\leq \frac{1}{3}$,

PP: only the trivial bound, error probability $\leq \frac{1}{2}$, that can be obtained by tossing a coin,

NP: no incorrect positive error, but $\Pr[M \text{ accepts } x]$ for $x \in A \subseteq$ NP can be arbitrarily small.

Definition 6.13. In addition to PP and BPP, the notion of error probability leads us to the following probabilistic complexity classes:

- $A \in$ RP (**r**andom **p**robabilistic **p**olynomial time), if there is a polynomial PTM M such that

$$x \in A \implies \Pr[M \text{ accepts } x] \geq \tfrac{2}{3} \text{ and}$$
$$x \notin A \implies \Pr[M \text{ accepts } x] = 0.$$

(no incorrect positive results).

- $A \in$ Co-RP $:\Longleftrightarrow \overline{A} \in$ RP, i.e., there is a polynomial PTM M such that

$$x \in A \Longrightarrow \Pr[M \text{ accepts } x] = 1 \text{ and}$$

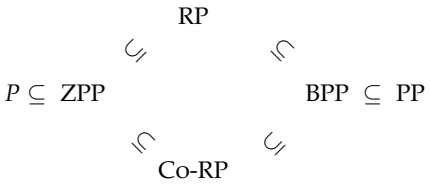
$$x \notin A \Longrightarrow \Pr[M \text{ accepts } x] \leq \tfrac{1}{3}.$$

  (no incorrect negative results).
- $A \in$ ZPP (**z**ero-error **p**robabilistic **p**olynomial time), if $A \in$ RP and $A \in$ Co-RP.

For the interpretation of ZPP, we consider a language $A \in$ ZPP. Then, there are polynomial PTM M^+ and M^- for $A \in$ RP and $\overline{A} \in$ RP. Consider now a PTM M that simulates the computations of M^+ and M^- in parallel and accepts the input if M^+ accepts it and rejects if M^- accepts it. In the case that M^+ rejects and M^- accepts, M returns "'don't know"'. Obviously, M is working error-free, i.e., the answers are always correct. It does, however, return an unsatisfying result with a probability of $\varepsilon \leq 1/3$. By repeating with independent random inputs, ε can be made arbitrarily small.

Example 6.14. The Rabin-Miller primality test (RM) shows that PRIMES $\in$ coRP. In 1987, Adleman and Huang have shown (the much more difficult result) that PRIMES is also in RP. Hence, PRIMES $\in$ ZPP.

Obviously, the following inclusions hold:



Furthermore, RP $\subseteq$ NP, Co-RP $\subseteq$ Co-NP and ZPP $\subseteq$ NP $\cap$ Co-NP.

Theorem 6.15. NP $\subseteq$ PP $\subseteq$ PSPACE.

Proof. Consider $A \in$ NP. By Lemma 6.11, there is a PTM M with $A = \{x \in \Sigma^* \ : \ \Pr[M \text{ accepts } x] > 0\}$. Let M' be a PTM accepting $(x, y_0 y_1 y_2 \dots)$ if, and only if, either $y_0 = 1$ or M accepts $(x, y_1 y_2 \dots)$.

Then,

$$\Pr[M' \text{ accepts } x] = \frac{1}{2} + \frac{1}{2}\Pr[M \text{ accepts } x],$$

and we obtain $A = \{x \ : \ \Pr[M' \text{ accepts } x] > \frac{1}{2}\} \in \text{PP}$.

On the other hand, consider $A \in \text{PP}$. Then,

$$x \in A \iff \Pr[M \text{ accepts } x] = \Pr_{y \in \{0,1\}^{p(n)}}[(x,y) \in L(M)] > \frac{1}{2}.$$

Therefore, for some given input x, all computations of M on input (x,y) with $y \in \{0,1\}^{p(n)}$ can be simulated using polynomial space to determine whether more than $2^{p(n)-1}$ of the pairs (x,y) are accepted.

Q.E.D.

Note that the relation between BPP and NP remains unclear.

In the following, we introduce a method to reduce the error probability of a BPP algorithm. Here, the fundamental idea is to use k iterations and then to decide for the most frequent result obtained.

Let M be a $p(n)$-time bounded PTM with an error probability $\leq \varepsilon < \frac{1}{2}$. Let M^k be a PTM accepting $(x, y_1 y_2 \dots y_k)$ with $y_i \in \{0,1\}^{p(n)}$ if and only if $|\{i \ : \ (x,y_i) \in L(M)\}| \geq k/2$. The algorithm M^k is polynomial if $k = k(n)$ is polynomial in n.

To compute the error probability of M^k, we need a result from probability theory.

Let $X_1, \dots, X_k$ be random variables over $\{0,1\}$ with $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for $0 < p < 1$ (Bernoulli random variables). The sum $X = \sum_{i=1}^{k} X_i$ is a binomially distributed random variable over $\mathbb{N}$. Its expectation is $E(X) = p \cdot k$. The following lemma gives a probability estimate for the case that the value of X differs less than d from the expectation:

Lemma 6.16 (Chernoff)**.** For $d \geq 0$,

$$\Pr[X - pk \geq d] \leq e^{-\frac{d^2}{4kp(1-p)}} \leq e^{-\frac{-d^2}{k}} \quad \text{and} \quad \Pr[pk - X \geq d] \leq e^{-\frac{-d^2}{k}}.$$

Coming back to our original problem, for $\bar{y} = y_1 \dots y_k$ (with $y_i \in$

$\{0,1\}^{p(n)}$), we define the random variables

$$X_i(\bar{y}) := \begin{cases} 1 & \text{if } (x, y_i) \in L(M), \\ 0 & \text{otherwise.} \end{cases}$$

Let A be a language that is decided by a BPP algorithm M with an error probability $\leq \varepsilon < \frac{1}{2}$. Then,

(1) For $x \notin A$, $p := \Pr[X_i = 1] \leq \varepsilon$. With $X := \sum_{i=1}^{k} X_i$,

$$\Pr[M^k \text{ accepts } x] = \Pr[X \geq \frac{k}{2}].$$

Applying Chernoff's Lemma, we obtain

$$\Pr[X \geq k/2] = \Pr[X - pk \geq k/2 - pk] \leq e^{-(\frac{1}{2}-p)^2 k} = 2^{-\Omega(k)}.$$

Let $q(n)$ be a suitable polynomial. For $k \geq c \cdot q(n)$ (c is a suitable constant), we obtain an incorrect positive error probability $\leq 2^{-q(n)}$.

An analogous statement holds for incorrect positive error probability:

(2) For $x \in A$, $\Pr[M \text{ accepts } x] = \Pr[X_i = 1] = p \geq 1 - \varepsilon$ and

$$\begin{aligned} \Pr[M^k \text{ does not accept } x] &= \Pr\left[X < \frac{k}{2}\right] \\ &= \Pr\left[pk - X \geq (p - \frac{1}{2})k\right] \\ &\leq e^{-(p-\frac{1}{2})^2 k} = 2^{-\Omega(k)}. \end{aligned}$$

Hence, we have shown:

Theorem 6.17. For every language $A \in \text{BPP}$ and every polynomial $q(n)$, there is a polynomial PTM M accepting A with an error probability $\leq 2^{-q(n)}$, i.e.,

$$\begin{aligned} x \in A &\implies \Pr[M^k \text{ accepts } x] \geq 1 - 2^{-q(|x|)}, \\ x \notin A &\implies \Pr[M^k \text{ accepts } x] \leq 2^{-q(|x|)}. \end{aligned}$$

An analogous statement also holds for the class RP.

From Theorem 6.17, we obtain an interesting result concerning the relationship between BPP and circuit complexity.

Let M be some BPP algorithm for A with an error probability of $\leq 2^{-q(n)}$. For all x,

$$\frac{|\{y \in \{0,1\}^{p(n)} \ : \ (x,y) \in L(M) \iff x \in A\}|}{2^{p(n)}} \geq 1 - 2^{-q(n)}.$$

It follows that for every fixed input length n, there are random values $y \in \{0,1\}^{p(n)}$ returning the correct result for *all* $x \in \Sigma^n$:

$$|\{y \in \{0,1\}^{p(n)} \ : \ y \text{ “bad” for at least one } x \in \Sigma^n\}| =$$
$$\sum_{|x|=n} |\{y \in \{0,1\}^{p(n)} \ : \ y \text{ “bad” for } x\}| \leq |\Sigma^n| \cdot 2^{-q(n)} \cdot 2^{p(n)}.$$

If $q(n)$ is chosen such that $\lim_{n\to\infty} |\Sigma^n| \cdot 2^{-q(n)} = 0$ (e.g., $q(n) = n^2$, or $q(n) = cn$ with $c \geq \log|\Sigma|$), we obtain that for large n at least one $y(n) \in \{0,1\}^{p(n)}$ gives the correct result for all $x \in \Sigma^n$;

Hence, there is a function $f : \mathbb{N} \to \{0,1\}^*$ with the following properties:

- f is polynomially-bounded: $|f(n)| = p(n)$ and
- for all sufficiently long $x \in \Sigma^*$,

$$x \in A \iff \underbrace{(x, f(x)) \in L(M)}_{\text{polynomial}}.$$

Definition 6.18. $A \in \Sigma^*$ is *non-uniform polynomially-decidable* ($A \in$ non-uniform P) if there is a function $f\colon \mathbb{N} \to \{0,1\}^*$ and a set $B \in \mathrm{P}$ such that

- $|f(n)| \leq p(n)$ for a polynomial p and
- $A = \{x \in \Sigma^* \ : \ (x, f(|x|)) \in B\}$.

Such a function f is called an *advice* function since it provides additional information $f(n)$ on every input length n that allows to decide A in polynomial time. Note that f itself does not need to be computable. The class non-uniform P is sometimes also denoted by

P/poly, "P with polynomial advice". This additional information $f(n)$ can be understood as the encoding of a polynomial circuit deciding A on input length n. Indeed, it is easy to see that

$$A \in \text{non-uniform P} \iff A \text{ is decided by a sequence of circuits of polynomial size.}$$

Corollary 6.19. BPP $\subseteq$ non-uniform P. Therefore, all problems in BPP are of polynomial circuit complexity.

Theorem 6.20. BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$.

Proof. It is sufficient to show that BPP $\subseteq \Sigma_2^p$. Since coBPP = BPP, it follows directly that BPP $\subseteq \Pi_2^p$.

Consider $A \in$ BPP. By Theorem 6.17, there is a polynomial PTM M deciding A with error probability $< 2^{-n}$:

$$\begin{aligned} x \in A &\implies \Pr_{y \in \{0,1\}^{p(n)}}[M \text{ accepts } x] > 1 - 2^{-n}, \text{ and} \\ x \notin A &\implies \Pr_{y \in \{0,1\}^{p(n)}}[M \text{ accepts } x] < 2^{-n}. \end{aligned}$$

In particular,

$$x \in A \iff |\{y \in \{0,1\}^{p(n)} \ : \ (x,y) \in L(M)\}| > 2^{p(n)}(1 - 2^{-n}).$$

Fix some x, $|x| = n$. Let $\Omega = \{0,1\}^{p(n)}$ and $B \subseteq \Omega$. We seek a criterion for the property $|B| > (1 - 2^{-n})|\Omega|$. The idea is to cover all of Ω with "few" images of B under translation modulo 2.

For $y, z \in \Omega$, let $y \oplus z := w_0 \dots w_{p(n)-1} \in \Omega$ with $w_i = y_i \oplus z_i$ (bitwise addition modulo 2). Let $B \oplus z := \{y \oplus z \ : \ y \in B\}$.

Lemma 6.21. For sufficiently large n and $B \in \{0,1\}^{p(n)}$ such that either

(i) $|B| < 2^{-n} \cdot 2^{p(n)}$ or
(ii) $|B| > (1 - 2^{-n}) \cdot 2^{p(n)}$

the following holds:

$$\text{(ii)} \iff \exists \overline{z} = (z_1, \dots, z_{p(n)}) \in \Omega^{p(n)} : \bigcup_i B \oplus z_i = \{0,1\}^{p(n)}.$$

Proof. ($\Rightarrow$): $\bigcup_i B \oplus z_i$ contains at most $p(n) \cdot |B|$ elements. If $\bigcup_i B \oplus z_i$ covers all of Ω, (i) is impossible since $p(n) \cdot 2^{-n}|\Omega| < |\Omega|$ for large n.

($\Leftarrow$): We use a probabilistic argument. Fix some $y \in \Omega$ and choose $z \in B \oplus y$. If we assume that (ii) holds, it follows that

$$\Pr_{z\in\Omega}[y \in B \oplus z] = \Pr_{z\in\Omega}[z \in B \oplus y] = \Pr_{z\in\Omega}[z \in B] > 1 - 2^{-n}.$$

Hence, we obtain:

$$\Pr_{\overline{z}\in\Omega^n}\left[\bigwedge_i y \notin B \oplus z_i\right] \leq \prod_i \Pr_{z_i\in\Omega}[y \notin B \oplus z_i] \leq 2^{-n\cdot p(n)}.$$

Therefore, the probability that some random $\overline{z} \in \Omega^n$ does *not* fulfil the conditions of the lemma can be approximated as follows:

$$\begin{aligned}\Pr_{\overline{z}\in\Omega^n}\left[\bigcup_i B \oplus z_i \neq \Omega\right] &\leq \sum_{y\in\Omega} \Pr_{\overline{z}\in\Omega^n}\left[\bigwedge_i y \notin B \oplus z_i\right] \\ &\leq 2^{p(n)} \cdot 2^{-n\cdot p(n)} < 1 \quad \text{for large } n.\end{aligned}$$

Hence, there must be a "good" $\overline{z}$. Q.E.D.

We can thus express A as follows: Let $B_x = \{y \in \Omega : (x, y) \in L(M)\}$. Then,

$$\begin{aligned} x \in A &\implies |B_x| > (1 - 2^{-n}) \cdot 2^{p(n)}, \text{ and} \\ x \notin A &\implies |B_x| < 2^{-n} \cdot 2^{p(n)}. \end{aligned}$$

Hence,

$$\begin{aligned} x \in A &\iff \exists \overline{z} \in \Omega^{p(n)} : \bigcup_{i=1}^{p(n)} B_x \oplus z_i = \Omega \\ &\iff \exists \overline{z} \in \Omega^{p(n)} \forall y \in \Omega \bigvee_{i=1}^{p(n)} \underbrace{y \in B_x \oplus z_i}_{\equiv y\oplus z_i \in B_x} \\ &\iff \exists \overline{z} \in \Omega^{p(n)} \forall y \in \Omega \bigvee_{i=1}^{p(n)} \underbrace{(x, y \oplus z_i) \in L(M)}_{\text{in P}}. \end{aligned}$$

Therefore, $A \in \Sigma_2^p$. Q.E.D.

6.3 Probabilistic proof systems and Arthur-Merlin games

We go back to the year 528 and turn our attention to the Court of King Arthur. A Round Table for 150 knights needs to be prepared. King Arthur is worried about peace at table as many knights are enemies. A seating arrangement needs to be found that makes sure no knights that are enemies are seated next to each other. Hence, King Arthur has the following problem: Given a graph $G = (P, E)$ with $P = \{\text{Arthur}\} \cup \{K_1 \dots K_{150}\}$, and $E = \{(x, y) \ : \ x \text{ not enemy with } y\}$; find a Hamilton cycle of G.

Arthur is a wise man and assumes that the design of such a seating arrangement might lead to evaluate all 150! possibilities for which the remaining time until the Round Table would not be sufficient. That is why he charges his magician Merlin with this task. Merlin possesses some super-natural power and can therefore find a peaceful arrangement if it does exist.

As most reasonable people, King Arthur does not completely rely on magic. He therefore always double-checks all solutions that Merlin proposes before actually implementing them. That is, once Merlin proposes a seating arrangement $k_0, k_1, \dots, k_{150}$ (let k_0 be the king), Arthur himself makes sure that for all j, $(k_j, k_{j+1}) \in E$.

However, the day comes when a new Round Table is going to take place. Some knights have reconciled, others have become enemies. Merlin finds out that there is no peaceful arrangement for this situation any more. King Arthur does not want to accept this result without proof, but a verification of all 150! possibilities is impossible.

Hence, Merlin needs to find a proof for the nonexistence of a seating arrangement that can be verified by Arthur. Since he cannot come up with one (as he does not know whether HAM $\in$ coNP), Merlin ends up in prison. After a while, the king regrets his impatience and is willing to accept a proof that he can verify with a probability of $1/2^{1000}$.

6.3.1 Interactive proof systems

The notion of a *proof* can—informally speaking—be defined as an interaction between a prover (P) and a verifier (V). After the interaction

is completed, the verifier decides whether to accept the proof. Hence, a *proof system* is a protocol defining the interaction of P and V on input x (the theorem to prove). As opposed to proof notion from classical logic, this approach allows interesting observations on complexity.

The class NP is characterised by the following *deterministic proof system*: A language $Q \subseteq \Sigma^*$ is in NP if there are Turing computable functions $P : \Sigma^* \to \Sigma^*$ and $V : \Sigma^* \times \Sigma^* \to \{\text{accept, reject}\}$ with

- V polynomial in the first argument (i.e., $\text{time}_V(x, y) \leq p(|x|)$ for some polynomial p).
- *Completeness:* for all $x \in \Sigma^*$, $x \in Q \implies V(x, P(x)) = \text{accept}$.
- *Correctness:* for all $P' : \Sigma^* \to \Sigma^*$ and all x, we have $x \notin Q \implies V(x, P'(x)) = \text{reject}$ (i.e., no prover P' can convince V of an *incorrect* proposition "$x \in Q$.")

Example 6.22. The graph isomorphism problem $\text{GRAPHISO} = \{(G, H) : G, H \text{ graphs}, G \cong H\} \in \text{NP}$. Without loss of generality, we can assume that $G = (V, E^G)$ and $H = (V, E^H)$. Then, on an input (G, H), there is the following proof system:

- P returns some permutation $\pi : V \to V$.
- V verifies whether $\pi : G \xrightarrow{\sim} H$.

It is unknown whether $\text{GRAPHISO} \in \text{coNP}$, that is, whether there is an NP proof system for $\text{GRAPHNONISO} = \{(G, H) : G \ncong H\}$.

In the following, we introduce *interactive proof systems* that allow a more sophisticated interaction between prover and verifier: the statements made by the prover are verified probabilistically in polynomial time; the verifier accepts with an error probability $\varepsilon > 0$.

Definition 6.23. Let Σ be an alphabet. An *interactive protocol* on Σ^* is a pair (P, V) of computable functions $P : \Sigma^* \to \Sigma^*$, $V : \Sigma^* \times \Sigma^* \times \{0,1\}^\omega \to \Sigma^* \cup \{\text{accept}, \text{reject}\}$ where V is polynomial in the first argument (i.e., $\text{time}_V(x, y, z) \leq p(|x|)$ for some polynomial p).

The *history* of (P, V) on $x \in \Sigma^*$ with the random word $y \in \{0,1\}^\omega$

is a sequence $U(x,y) = u_0, u_1, \ldots$ with $u_0 = x$ and $u_{i+1} = u_i a_i$ where

$$a_i := \begin{cases} V(x, u_i, y) & \text{if } i \text{ is even,} \\ P(u_i) & \text{if } i \text{ is odd and } a_i \notin \{\text{accept}, \text{reject}\}. \end{cases}$$

We say (P,V) accepts x (with the random word y) if $u_k = \text{accept}$ for some k. For every $x \in \Sigma^*$, the history of (P,V) is a random variable $U(x) : y \mapsto U(x,y)$.

Definition 6.24. Consider $Q \subseteq \Sigma^*$. An *interactive proof system* for Q is an interactive protocol (P,V) satisfying the following requirements:

- *Completeness:* for all $x \in \Sigma^*$,
  $x \in Q \Longrightarrow \Pr[(P,V) \text{ accepts } x] > \frac{2}{3}$.
- *Correctness:* for all $P' : \Sigma^* \to \Sigma^*$ and all x,
  $x \notin Q \Longrightarrow \Pr[(P',V) \text{ accepts } x] < \frac{1}{3}$.

A *round* of $U(x,y)$ is a pair (a_{2i}, a_{2i+1}) (hence, a "message" of P followed by an "answer" of V). We say a protocol has $\leq q$ rounds if, for all $x \in \Sigma^*$ and all $y \in \{0,1\}^\omega$, the history $U(x,y)$ has at most $q(|x|)$ rounds.

Definition 6.25. IP (respectively, IP$[q]$) is the class of all $Q \subseteq \Sigma^*$ for which there is an interactive proof system (respectively, one with $\leq q$ rounds).

Example 6.26. GRAPHNONISO $\in$ IP[2]. The proof system (P,V) works on input (G_0, G_1) with $G_i = (V_i, E_i)$ as follows:

- V chooses some index $i, i' \in \{0,1\}$ at random and some permutation $\pi, \pi' : V \to V$. Then, he computes $H = \pi G_i$, $H' = \pi' G_i$ and sends both H and H' to the prover. (The chosen indices i, i', and permutations π, π' remain secret!)
- P replies with $j, j' \in \{0,1\}$ such that $H \cong G_j$, and $H' \cong G_{j'}$.
- V accepts if $i = j$ and $i' = j'$; otherwise V rejects.

Analysis.

- If $G_0 \ncong G_1$, then H and H' are isomorphic to exactly one input graph. Thus, P can determine i and i'. It follows $G_0 \ncong G_1 \Longrightarrow \Pr[(P,V) \text{ accepts } (G_0, G_1)] = 1$.

- If $G_0 \cong G_1$ then H and H' are isomorphic to both graphs and all $G \cong G_0, G_1$ appear with equal probability. Every prover P' can do no better than guessing j, j'. Hence, for all P': $G_0 \cong G_1 \Longrightarrow \Pr[(P, V) \text{ accepts } (G_0, G_1)] \leq \frac{1}{4}$.

Lemma 6.27. NP $\subseteq$ IP[1] $\subseteq$ IP[2] $\subseteq \cdots \subseteq$ IP $\subseteq$ PSPACE.

*Proof (*IP $\subseteq$ PSPACE*).*
Let (P, V) be an interactive proof system for Q. Since V is polynomial in the first argument, V on input (x, u_i, y) reads only the first $p(|x|)$ symbols of u_i and y. It is possible, using polynomial space, to simulate all possible histories $U(x, y)$ and to determine $\Pr[(P, V) \text{ accepts } x]$. Q.E.D.

It is interesting to know where—between NP and PSPACE—the class of interactively provable languages is located. For this, we return to Arthur and Merlin.

Definition 6.28. An *Arthur-Merlin game* (for a language Q) is an interactive proof system for Q with the additional requirement that the prover (Merlin) can see the random bits used by the verifier (Arthur). Without loss of generality, the messages from Arthur to Merlin are then composed of a sequence $y_1, \ldots, y_r$ $(r \leq p(|x|))$ of random bits.

AM (respectively, AM[q]) is the class of all $Q \subseteq \Sigma^*$ that have an Arthur-Merlin game (respectively, one with $\leq q(|x|)$ rounds).

Obviously, AM $\subseteq$ IP, and
AM[q] $\subseteq$ IP[q].

In the following, we see that one of the most difficult problems from PSPACE is in AM.

6.3.2 An Arthur-Merlin game for the permanent of a matrix

Definition 6.29. Let R be a ring, $R' \subset R$ some subset and $A \in M_n(R')$ an $n \times n$ matrix over R'. The *permanent* of A (over R) is defined as

$$\operatorname{per}_R A = \sum_{\sigma \in S_n} a_{1\sigma(1)} \ldots a_{n\sigma(n)}.$$

The analogy to determinants is striking:

$$\det A = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) a_{1\sigma(1)} \dots a_{n\sigma(n)}.$$

By removing the ith line and the jth column, one obtains the ij minor A_{ij} of A, and it follows

$$\det A = \sum_{i=1}^{n} (-1)^{i+1} a_{i1} \cdot A_{i1} \quad \text{and}$$
$$\operatorname{per} A = \sum_{i=1}^{n} a_{i1} \cdot A_{i1}.$$

However, the determinant can be computed efficiently where as no efficient algorithm is known for the permanent. It is currently assumed that in general it is not efficiently computable. The following result confirms this assumption.

Definition 6.30. #P is the class of all functions $f : \Sigma^* \to \mathbb{N}$ for which there is a polynomial nondeterministic TM M such that the number of accepting computations of M on x is exactly $f(x)$.

Theorem 6.31 (Toda)**.** PH $\subseteq$ #P.

Theorem 6.32 (Valiant)**.** The permanent (over $\mathbb{Z}$) of 0-1-matrices is #P -complete.

A polynomial algorithm to compute the permanent would therefore imply PH = P. An interactive proof system for

$$\text{PER} = \{(A, q) \ : \ A \in M_n(\{0,1\}), \operatorname{per}_{\mathbb{Z}} A = q\}$$

hence implies that every problem $Q \in$ PH has an interactive proof system. This would in particular be true for

$$\overline{\text{HAM}} = \{G \ : \ G \text{ contains no Hamilton circle } \} \in \text{coNP} \subseteq \text{PH}.$$

Theorem 6.33. There is an Arthur-Merlin game for PER.

Proof. (1) Since $A \in M_n(\{0,1\}) \Longrightarrow 0 \leq \operatorname{per}_{\mathbb{Z}}(A) \leq n!$. Let $p > n!$ be a prime number. Then, $\operatorname{per}_{\mathbb{F}_p}(A) = \operatorname{per}_{\mathbb{Z}}(A)$ over the field $\mathbb{F}_p$.

(2) For a field $\mathbb{F}$, let $\mathbb{F}[X]_d = \{f \in \mathbb{F} : \text{degree} f \leq d\}$. If $A \in M_n(\mathbb{F}[X]_d)$ then $\text{per} A \in \mathbb{F}[X]_{nd}$.

(3) *The protocol:*
Arthur and Merlin work with a list $L = \{(A_1, q_1) \dots (A_r, q_r)\}$ where $A_i \in M_k(\mathbb{F}_p)$ and $q_i \in \mathbb{F}_p$ $(k \leq n)$. L is correct if $\text{per} A_i = q_i$ for $i = 1, \dots, r$.
Beginning: $L = \{(A, q)\}$, $A \in M_n(\mathbb{F}_p)\}$.
In a sequence of subprotocols 'expand' and 'reduce', L is changed until, at the
End: $L = \{(B, s)\}$, $B \in M_2(\mathbb{F}_p)$.
Arthur accepts if, and only if, $\text{per} B = s$.

- if L contains only one pair: $L = \{(A, q)\}$, $A \in M_k(\mathbb{F}_p)\}$, $k > 2$:

  Expansion step:
  $L \mapsto L' = \{(A_1, q_1), \dots, (A_k, q_k)\}$ where $A_i \in M_{k-1}(\mathbb{F})_p$.
  Subprotocol **expand**(L)
  Input: $L = \{(A, q)\}$
  Merlin: computes $q_i = \text{per}(A_{i1})$ for $i = 1, \dots, k$ and sends the results $q_1, \dots, q_k$ to Arthur.
  Arthur: verifies whether $\sum_{i=1}^{k} a_{i1} q_i = q$.
  If not, he rejects;
  otherwise, he sets $L' = \{(A_{i1}, q_1), \dots, (A_{k1}, q_k)\}$.

  For this step, we have

  $$L \text{ correct} \Longrightarrow L' \text{ correct.}$$
  $$L \text{ incorrect} \Longrightarrow L' \text{ incorrect (no matter how Merlin plays).}$$

- if $|L| > 1$,
  Reduction step: $L \mapsto L'$ with $|L'| = |L| - 1$ such that

  $$L \text{ correct} \Longrightarrow L' \text{ correct.}$$
  $$L \text{ incorrect} \Longrightarrow L' \text{ incorrect with high probability.}$$

  Consider $(A, q_1), (B, q_2) \in L$, $A, B \in M_k(\mathbb{F}_p)$. Set

  $$C(X) = (1 - X)A + XB$$

$$= \begin{pmatrix} & \vdots & \\ \dots & \alpha x + \beta & \dots \\ & \vdots & \end{pmatrix} \in M_k(\mathbb{F}_p[X]).$$

With per $C(X) =: f \in \mathbb{F}_p[X]$, we have

$$\begin{aligned} C(0) = A, &\quad \text{hence} \quad f(0) = \text{per } A; \\ C(1) = B, &\quad \text{hence} \quad f(1) = \text{per } A. \end{aligned}$$

Remark: To verify whether per $A = q_1$ and per $B = q_2$, it therefore suffices to determine the polynomial f and to evaluate it at 0 and 1.

Subprotocol **reduce**(L):
Input: $\{(A, q_1), (B, q_2)\}, \quad A, B \in M_k\mathbb{F}_p$.
Merlin: sends Arthur $c_0, \dots, c_k \in \mathbb{F}_p$
(claiming that $f(X) = c_0 + c_1X + \cdots + c_kX^k$).
Arthur: sets $g(X) = c_0 + c_1X + \cdots + c_kX^k$, and
verifies whether $g(0) = q_1$ and $g(1) = q_2$.
If not, he rejects;
otherwise, he chooses a random number $a \in \mathbb{F}_p$ and
sets $L' = (L - \{(A, q_1), (B, q_1)\} \cup \{(C(a), g(a))\}$.

For this step, we have

L correct $\Longrightarrow$ Merlin can play in such a way that L' is correct by sending the correct coefficients of f, i.e., such that $g(X) = f(X) = \text{per } C(X)$, and in particular $g(a) = \text{per } C(a)$.

L incorrect $\Longrightarrow$ with high probability L' incorrect.

Reason: Assume that per $A = \text{per } C(0) \neq q_1$. Merlin sends incorrect coefficients since $g(0) = q_1 \neq f(0) = \text{per } A$. Hence, we have

$$f \neq g \Longrightarrow |\{a \ : \ f(a) = g(a)\}| \leq k$$
$$\Longrightarrow \Pr[\text{per } C(a) = g(a)] \leq \frac{k}{p} < \frac{1}{(n-1)!}$$

for $p > n!$, and $k \leq n$.

Hence, we obtain the Arthur-Merlin game described in Algorithm 6.2.
Analysis.

(a) The game is indeed an Arthur-Merlin protocol, i.e., all computations by Arthur are in P since

- $p < 2n!$, $|p| = O(n \log n)$ since $n! = 2^{O(n \log n)}$,
- the arithmetical operations in $\mathbb{F}_p$ are polynomial in $|p|$ and
- the protocol uses $n-2$ expansion steps and $\sum_{i=2}^{n-2}(i-1) = \frac{(n-1)(n-2)}{2}$ reduction steps.

(b) $(A,q) \in$ PER $\Longrightarrow \Pr[(M,A) \text{ accepts } (A,q)] = 1$.

Algorithm 6.2. Arthur-Merlin game for PER

Input: (A,q), $A \in M_n(\{0,1\}), q \in \mathbb{N}$
Merlin:
sends Arthur a prime number $p \in [n!, 2n!]$ together with a short proof showing that p is prime[a].
Arthur:
verifies that p is indeed a prime number between $n!$ and $2n!$.
if *p not prime* **then reject**

```
/* For the remainder of the protocol, all calculations are
   done in F_p.                                              */
```

$L := \{(A,q)\}$
while $L \neq \{(B,s)\}$ *for a* $B \in M_2(\mathbb{F}_p)$ **do**
if $|L| = 1$ **then expand**(L) **else reduce**(L)
endwhile
Arthur:
verifies whether per $B = s$.
if *yes* **then** accept **else** reject

---

[a] It is known that for all $a \in \mathbb{N}$ there is a prime number between a and $2a$ (Bertrand's postulat). Since PRIMES $\in$ NP, there are short proofs for the fact that p is prime (i.e., there is an $L \in$ P with PRIMES $= \{p \ : \ \exists w |w| \leq |p|^k \ : \ (p,w) \in L\}$ (where w is a short proof).

(c) Let $(A,q) \notin$ PER and M' some prover. Then, (M',A) accepts $(A,q) \Longrightarrow M'$ has cheated successfully in at least one reduction step. This can occur in one single reduction step with a probability of at most $\frac{1}{(n-1)!}$. Altogether, we obtain

$$\begin{aligned}\Pr[(M'A) \text{ accepts } (A,q)] &< 1 - \left(1 - \frac{1}{(n-1)!}\right)^{\frac{(n-1)(n-2)}{2}} \\ &< \frac{(n-1)(n-2)}{(n-1)!} = \frac{1}{(n-3)!}. \quad \text{Q.E.D.}\end{aligned}$$

In 1992, this theorem was published by Lund, Fortnow, Karloff and Nisan in JACM 39(4).

6.3.3 IP = PSPACE

Only one month after Lund, Fortnow, Karloff and Nisan, Shamir showed that IP = PSPACE. In order to do so, he constructed an interactive proof system (even an Arthur-Merlin game) for QBF. QBF is the problem to evaluate quantified Boolean formulae, and it is PSPACE-complete. An Arthur-Merlin game for this problem therefore suffices to show that all $Q \in$ PSPACE have an interactive proof system (and even an Arthur-Merlin game). At the same time it shows that Arthur-Merlin games are equally strong as interactive proof systems.

Theorem 6.34 (Shamir)**.** There is an Arthur-Merlin game for QBF.

Proof (Simplified version).

(1) Arithmetisation of a formula from quantified propositional logic. First, let $\varphi(X_1,\ldots,X_n)$ be a propositional formula (without quantifiers) and let $\mathbb{F}$ be some arbitrary field. A map $\varphi \mapsto F_\varphi \in \mathbb{F}[X_1,\ldots,X_n]$ is defined inductively as follows:

$$\begin{aligned}F_{X_i} &= X_i, \\ F_{\alpha\wedge\beta} &= F_\alpha \cdot F_\beta, \\ F_{\neg\alpha} &= 1 - F_\alpha, \\ F_{\alpha\vee\beta} &= F_\alpha \circ F_\beta := F_\alpha + F_\beta - F_\alpha F_\beta.\end{aligned}$$

Let $\mathfrak{I} : \{X_1, \ldots, X_n\} \mapsto \{0,1\}$ be an interpretation with $\mathfrak{I}(X_1) = \varepsilon_1, \ldots, \varepsilon(X_n) = \varepsilon_n$. We write $\varphi(\varepsilon_1, \ldots, \varepsilon_n)$ for $\mathfrak{I}(\varphi)$. For every field $\mathbb{F}$ and all $\varepsilon_1, \ldots, \varepsilon_n \in \{0,1\}$, $F_\varphi(\varepsilon_1, \ldots, \varepsilon_n) = \varphi(\varepsilon_1, \ldots, \varepsilon_n)$.
For $f \in \mathbb{F}[Y, \overline{X}]$, we define the following:

$$\begin{aligned}
(\forall Y f)(\overline{X}) &= f(0, \overline{X}) \cdot f(1, \overline{X}) && \in \mathbb{F}[\overline{X}], \\
(\exists Y f)(\overline{X}) &= f(0, \overline{X}) \circ f(1, \overline{X}) && \in \mathbb{F}[\overline{X}], \\
(RYf)(\overline{X}) &= f \pmod{Y^2 - Y} && \in \mathbb{F}[Y, \overline{X}].
\end{aligned}$$

(all Y^i with $i > 0$ are replaced by Y)

Hence, we obtain the following arithmetisation of quantified propositional formulae with quantifiers:

$$\begin{aligned}
F_{\forall Y \varphi} &= (\forall Y F_\varphi), \\
F_{\exists Y \varphi} &= (\exists Y F_\varphi).
\end{aligned}$$

Obviously, for all quantified propositional formulae $\Psi(X_1, \ldots, X_k)$, we have $F_\Psi(\varepsilon_1, \ldots, \varepsilon_k) = \Psi(\varepsilon_1, \ldots, \varepsilon_k)$. In particular if $\mathrm{free}(\Psi) = \varnothing$, $\Psi \in \textsc{qbf} \iff F_\Psi = 1$ holds.
However, we have the following problem: The explicit construction of F_Ψ is just as difficult as the evaluation of the QBF formula Ψ. Length and degree of the polynomial F_Ψ can become arbitrarily large since every application of the quantifiers $\forall$ and $\exists$ double both the length and degree.

(2) Degree reduction using R.
For $u \in \{0,1\}$, we have $(RYf)(u, \overline{X}) = f(u, \overline{X})$. Further, for $f \in \mathbb{F}[X_1, \ldots, X_n]$ set

$$(R^* f)(X_1, \ldots X_n) = (RX_1 RX_2 \ldots RX_n f)(X_1, \ldots, X_n).$$

For $\varepsilon_1, \ldots \varepsilon_n \in \{0,1\}$,

- $(R^* f)(\varepsilon_1, \ldots \varepsilon_n) = f(\varepsilon_1, \ldots \varepsilon_n)$, and
- $\mathrm{degree}(R^* f) \leq n$.

Let $\Psi = Q_1 X_1 \ldots Q_n X_n \varphi(X_1, \ldots, X_n)$ be a quantified Boolean formula with $\mathrm{free}(\Psi) = \varnothing$. Then, $\Psi \in \textsc{qbf}$ if, and only if,

$(Q_1X_1R^*Q_2X_2\ldots R^*Q_nX_nR^*F_\varphi) = 1$ since the R^* operator leaves the functional values invariant for the arguments 0,1.

(3) Arthur-Merlin game.

Let $f \in \mathbb{F}[X_1,\ldots,X_n]$, $\mathbb{F}$ be finite. We assume there is an Arthur-Merlin game with $(M,A)_f$ for $\{(u,v) \in \mathbb{F}^{n+1} \ : \ f(\overline{u}) = v\}$ with

(i) $f(\overline{u} = v) \Longrightarrow \Pr[M \text{ accepts } (\overline{u},v)] = 1$,

(ii) $f(\overline{u} \neq v) \Longrightarrow \Pr[M' \text{ accepts } (\overline{u},v)] < \varepsilon$ for all M'.

Let $g \in \{(\exists X_i f),(\forall X_i f),(RX_i f) \ : \ i = 1,\ldots,n\}$. We assume Arthur knows a d with degree $g \leq d$.

Based on these assumptions, we construct an Arthur-Merlin game $(M,A)_g$ that calls $(M,A)_f$ exactly once as subprotocol with

(i) $f(\overline{u} = v) \Longrightarrow \Pr[M \text{ accepts } (\overline{u},v)] = 1$,

(ii) $f(\overline{u} \neq v) \Longrightarrow \Pr[M' \text{ accepts } (\overline{u},v)] < \varepsilon + \frac{d}{|\mathbb{F}|}$ for all M'.

We obtain the following different cases:

(a) $g(\overline{X}) = (\forall Yf)(Y,\overline{X})$.
Merlin wants to show that $g(\overline{u}) = v$. He sends the coefficients of a polynomial $s(Y)$ (requiring that $s(Y) = f(Y,\overline{u})$). If degree $s > d$ or $s(0)\cdot s(1) \neq v$, Arthur rejects. Otherwise, Arthur chooses some random $w \in \mathbb{F}$. Merlin now needs to convince Arthur with the protocol $(M,A)_f$ that $f(w,\overline{u}) = s(w)$.

(b) $g(\overline{X}) = (\exists Yf)(Y,\overline{X})$.
Analogously, with $s(0)\circ s(1)$ instead of $s(0)\cdot s(1)$.

(c) $g(Y,\overline{X}) = (RYf)(Y,\overline{X})$.

We use

Lemma 6.35. $(RYf)(Y,\overline{X}) = f(0,\overline{X}) + [f(1,\overline{X}) - f(0,\overline{X})]\cdot Y$.

Proof. Let $f = \sum_{i=0}^{m} Y^i \cdot g_i(\overline{X})$. Then,

$$f(0,\overline{X}) = g_0(\overline{X}) \quad \text{and} \quad f(1,\overline{X}) = \sum_{i=0}^{m} g_i(\overline{X}).$$

Hence, we obtain $(RYf) = g_0(\overline{X}) + \sum_{i=0}^{m} g_i(\overline{X}) = f(0,\overline{X}) + [f(1,\overline{X}) - f(0,\overline{X})]\cdot Y$. Q.E.D.

Merlin wants to convince Arthur that $g(z, \overline{u}) = v$. He sends Arthur the coefficients of a polynomial $s(Y)$ (requiring that $s(Y) = f(Y, \overline{u})$). If degree $s > d$ or $s(0) + z(s(1) - s(0)) \neq v$, Arthur rejects. Otherwise, he sends Merlin some random $w \in \mathbb{F}$. Merlin now needs to convince Arthur with the protocol $(M, A)_f$ that $f(w, \overline{u}) = s(w)$.

The game described above fulfils the completeness requirement. As for correctness, we note that M' has the following possibilities to cheat successfully:

- In $(M, A)_f$ (with probability ε).
- If $s(Y) \neq f(Y, \overline{u})$ correspond at a randomly selected point w (probability $\leq \frac{d}{\mathbb{F}}$).

(4) Summary.

Given $\Psi = Q_1 X_1 \dots Q_n X_n \varphi$. Merlin convinces Arthur that

$$G_\Psi = Q_1 X_1 R^* \dots R^* Q_n X_n R^* F_\varphi = 1$$

with the help of the above-mentioned reduction steps. At the end, an equation $F_\varphi(u_1, \dots, u_n) = v$ needs to be verified. For a propositional formula φ, this is possible in polynomial time.
The error probability of the complete protocol is at most

$$\frac{\#\exists, \forall, R\text{-operators} \cdot \text{ maximal degree}}{|\mathbb{F}|} = \frac{O(n^2) + O(|\varphi|^2}{|\mathbb{F}|}.$$

Therefore, it suffices to choose a field $\mathbb{F}_p$ with $p \geq c \cdot |\varphi|^4$. Q.E.D.