Complexity Theory

Carsten Lutz

Winter semester 2006/2007

Contents

1 Introduction **1**

2 Turing Machines **2**

2.1 Deterministic Turing Machines 2
2.2 Multiple Tapes 5
2.3 Nondeterministic Turing Machines 8
2.4 DTM vs. NTM 9

3 Complexity Classes **10**

4 Tractable versus Intractable **15**

4.1 Examples 15
4.2 The Speedup Theorem 17
4.3 The Hierarchy Theorem 19
4.4 Basics of NP 21
4.5 SAT: A Natural NP-complete Problem 23

5 More NP-complete Problems **25**

5.1 P vs. NP 27
5.2 The Complement of NP 29

6 The Polynomial Hierarchy **30**

7 PSpace **33**

1 Introduction

The subject of complexity theory is to study the computational properties of problems, most notably *how hard* it is to compute a solution to (an instance of) a problem. Hardness is measured in terms of the resources (mainly time and memory) that are required by algorithms that solve the problem.

To see what a problem is, let us consider three examples.

- *Graph reachability.* A *directed graph* $G = (V, E)$ consists of a set of *nodes* V and a set of *edges* $E \subseteq V \times V$. The graph reachability problem is to decide, given a graph G and two nodes $v, v' \in V$, whether v' is reachable from v by travelling along the edges E.
- *Propositional satisfiability.* Given a formula φ of propositional logic, decide whether there is a truth assignment that makes the formula true.
- *Universality of regular expressions.* Given a regular expression π defining a language $L(\pi)$ over alphabet Σ, is $L(\pi) = \Sigma^*$?

We will see later that the computational hardness of these problems is quite different. As illustrated by the second problem, complexity theory is a very important subject for *computational* logic.

To set the stage for this lecture, let us discuss some fundamental issues concerning problems, algorithms, and their complexity:

Complexity of problem vs. complexity of algorithm. Usually, there are many different algorithms for solving the same problem. Some of them may be efficient w.r.t. a given resource, some not. Consider for example graph reachability and the following algorithm:

> Generate all possible sequences $v_0, \ldots, v_n$ of nodes of length at most $|V|$ (the number of nodes). For each sequence, check whether it starts with v, ends with v', and is such that $(v_i, v_{i+1}) \in E$ for all $i < n$.

Regarding temporal resources, this algorithm needs $\Sigma_{i=1..|V|}(|V|^i \cdot i)$ steps to compute a solution. For a graph with only 100 nodes, these are more than 10^{200} steps. Compare this to the number of atoms that are believed to exist in the universe: $\approx 10^{79}$. We will later see that there are *much* faster algorithms for graph reachability.

Complexity theory is *not* about the complexity of concrete algorithms, since they may be wasteful with resources like the algorithm above. It is about the complexity of problems, i.e., we are interested in the *best possible* algorithm for a given problem.

Number of instances. We will only consider problems that have an infinite number of instances. Indeed, problems with only a finite number of instances are not interesting for complexity theory because they always admit an algorithm that simply perform a lookup in a table storing the answers for all instances. It may be difficult to generate the table, but still there *always* exists such a (completely uninteresting) algorithm.

Undecidability. There are undecidable problems that cannot be solved by *any* algorithm. Although such problems do play a role in computational logic, in this lecture we will only be concerned with decidable problems.

Decision problems vs. function problems. Every problem in the above list has only "yes" and "no" as a possible answer. Such problems are called *decision problems*, other problems are called *function problems.* The following is an example of a function problem:

- *Minimal assignment.* Given a formula φ of propositional logic, return a truth assignment that satisfies φ and makes only a minimal number of variables true.

Usually, function problems can be converted into a roughly equivalent decision problem by providing a *target value.* In the case of problem above, we add a non-negative integer k as an additional input and ask whether there is a truth assignment that satisfies φ and makes at most k variables true. Observe that an algorithm for the function problem can easily be converted into an algorithm for the decision problem.

There is a branch of complexity theory which treats function problems as first class citizens. In this lecture, we will only consider decision problems.

2 Turing Machines

In complexity theory, problems are usually formalized as formal languages and algorithms are Turing machines. By the *Church-Turing Thesis*, the problems computable by a Turing machine are precisely those problems that are computable in the intuitive sense. Though such a claim cannot be fornally proved, Turing machines have indeed turned out to be equivalent to many other models of computation used in theory (such as Church's λ-calculus and register machines) and in practice (such as most programming languages). The advantage of Turing machines is that they are extremely simple. In the words of Papadimitriou, *it is amazing how little we need to have everything.*

Still, Turing machines are merely a technical tool that could be replaced by a different tool without essentially changing the obtained results. What we study is *not* Turing machines, but the nature of efficient computation (which is much more fundamental).

2.1 Deterministic Turing Machines

A Turing machine (TM) is a machine that has a finite control and an infinite tape to work on. For this lecture, we assume that the tape is bounded to the left and unbounded to the right.

Definition 1. A *(deterministic) Turing machine* is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, where

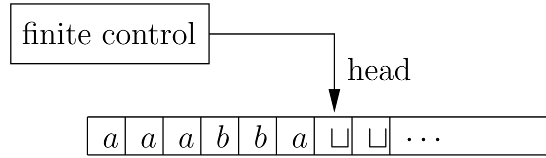
- Q is a finite set of *states* such that $\{q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}\} \subseteq Q$;

- Σ is the finite *input alphabet* such that $\sqcup \notin \Sigma$;
- Γ is the finite *tape alphabet* such that $\Sigma \cup \{\sqcup\} \subseteq \Gamma$;
- $q_0 \in Q$ is the *initial state*;
- $q_{\mathsf{acc}} \in Q$ is the *accepting state*;
- $q_{\mathsf{rej}} \in Q$ is the *rejecting state*;
- $\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the (total) *transition function*.

$\triangle$

The input to a TM is written left-boundedly on the tape and the TM starts in an initial state with the head on the left-most tape cell. The remainder of the tape is filled with the blank symbol $\sqcup$. Then, $\delta(q, a) = (q', b, L/R)$ means that when M is in state q and a is the symbol underneath the head, then M replaces a with b, moves the head left/right, and switches to state q'. If the head is on the left-most cell and the TM needs to execute $\delta(q, a) = (q', b, L)$, then the symbol a is replaced with b, the state is switched to q', the TM stays on the left-most cell, and the computation continues normally.

The overall current state of a TM can be described by a *configuration*, which is a word uqv with $u, v \in \Gamma^*$ and $q \in Q$. The configuration uqv means that the tape content is uv followed by infinitely many $\sqcup$ symbols, the TM is in state q, and the head is on the left-most symbol of v.



A configuration uqv is *accepting* if $q = q_{\mathsf{acc}}$ and *rejecting* if $q = q_{\mathsf{rej}}$. For every configuration c that is neither accepting nor rejecting, the transition function defines a unique successor configuration c', written $c \vdash_M c'$:

- if $\delta(q, a) = (q', b, L)$, then for all $u, v \in \Gamma*$,
  - $ua'qav \vdash_M uq'a'bv$
  - $qav \vdash_M q'bv$ (special case left-most tape cell);
- if $\delta(q, a) = (q', b, R)$, then for all $u, v \in \Gamma*$,
  - $uqav \vdash_M ubq'v$
  - $uq \vdash_M ubq'$ if $a = \sqcup$.

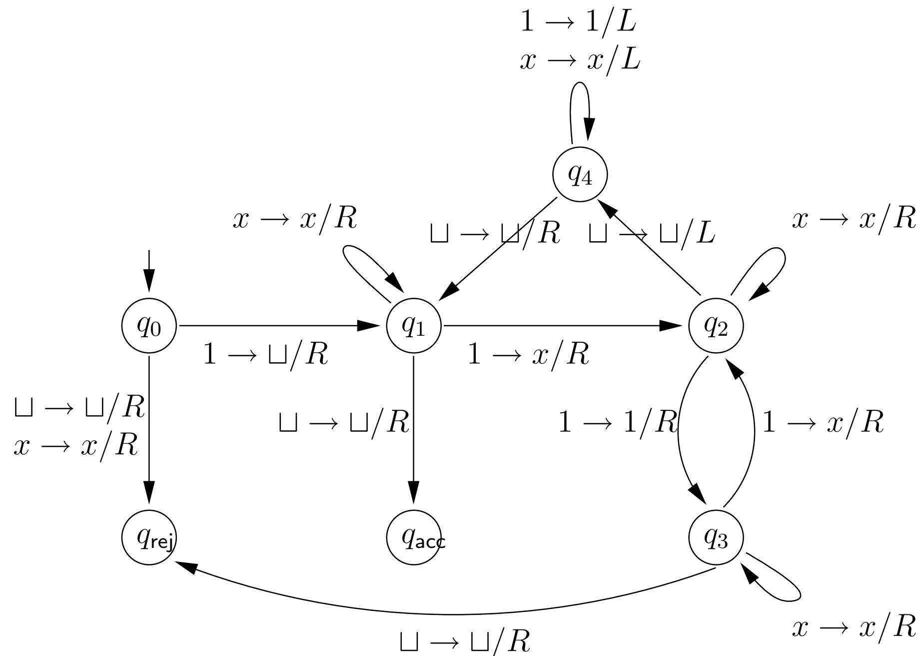
A finite or infinite sequence $c_0 \vdash_M c_1 \vdash_M \cdots$ is called a *computation* of M. The computation is *on* w if $c_0 = q_0 w$ and it is *accepting* if it is finite and its last configuration is accepting. A TM *accepts* an input $w \in \Sigma^*$ if its unique computation on w is accepting.

As an example, we define a TM which works over the singleton input alphabet $\Sigma = \{1\}$, stops on every input, and accepts input w if the length of w is a power of two. Intuitively, the TM does the following:

- sweep right over the whole input, replacing every second 1 with x;
- accept if only a single 1 was found;
- reject if this is not the case and the number of 1's found is odd (i.e., the last 1 seen was skipped rather than replaced with x);
- otherwise, move head back to the left end of tape;
- repeat sweeping process as long as possible, replacing every second 1 and ignoring the existing x.

More precisely, $M := (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, where

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$;
- $\Sigma = \{1\}$;
- $\Gamma = \{1, x, \sqcup\}$;
- δ is described in terms of the following state diagram:



Observe that δ is total and a function, i.e., in the state diagram we need exactly one edge for every state and every element of Γ.

Here is a sample computation of M on input 1111:

$$
\begin{array}{lll}
q_0 1111 & \sqcup q_4 x 1 x \sqcup & \sqcup x q_4 x x \sqcup \\
\sqcup q_1 111 & q_4 \sqcup x 1 x \sqcup & \sqcup q_4 x x x \sqcup \\
\sqcup x q_2 11 & \sqcup q_1 x 1 x \sqcup & q_4 \sqcup x x x \sqcup \\
\sqcup x 1 q_3 1 & \sqcup x q_1 1 x \sqcup & \sqcup q_1 x x x \sqcup \\
\sqcup x 1 x q_2 \sqcup & \sqcup x x q_2 x \sqcup & \sqcup x q_1 x x \sqcup \\
\sqcup x 1 q_4 x \sqcup & \sqcup x x x q_2 \sqcup & \sqcup x x q_1 x \sqcup \\
\sqcup x q_4 1 x \sqcup & \sqcup x x q_4 x \sqcup & \sqcup x x x q_1 \sqcup \\
 & & \sqcup x x x \sqcup q_{\mathsf{acc}}
\end{array}
$$

Intuitively, this TM solves the decision problem asking whether a number given in unary is a power of two. It stops on every input and thus partitions the set Σ^* into two parts: those numbers that are a power of two (the "yes" instances of the problem) and those that are not (the "no" instances of the problem).

This example suggests to view a problem as a formal language, i.e., as a (usually infinite) set of finite words over some alphabet Σ. Formally, we define the language *accepted* by a TM M as

$$L(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

In the following, we use the word "problem" as a synonym for a formal language.

Observe that there are two possible reasons for a word *not* to be in $L(M)$: M may stop in the rejecting state or it may not stop at all. Obviously, only TMs that stop on every input actually *solve* a decision problem and such TMs best correspond to our intuition of a (terminating) algorithm. Formally, we say that a TM *decides* a language L if $L(M) = L$ and L stops on every input.

Is a TM a *realistic* model of computation? With its infinite tape and basic instruction set, a TM may seem very different from a modern computer. On the other hand,

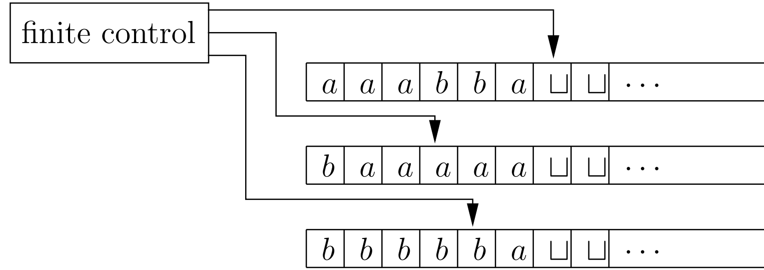
1. it is not difficult to see that more sophisticated instructions can be formulated in terms of the basic ones; e.g., if the word on the string is a binary encoding of a natural number, incrementation, addition, and other arithmetic operations are easily carried out by a TM; the number of basic instructions needed to carry out such operations are not fundamentally different from the number of instructions needed by a real computer;

2. when programming a computer, we think of it as having infinite memory, which is justified by mechanisms such as paging, swapping, etc.

In this sense, a TM is quite realistic if we assume a classical model of computation.

2.2 Multiple Tapes

When we define complexity classes, we will also allow TMs that have more than a single tape. There is one head for every tape and the positions of different heads need not agree:



A *k-tape TM* is a TM with k tapes. To describe k-tape TMs formally, we use a transition function of the form

$$\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

Intuitively,

$$\delta(q, a_1, \ldots, a_k) = (q', b_1, \ldots, b_k, M_1, \ldots, M_k)$$

means that if the TM is in state q and reads a_i on the i-th tape for all i with $1 \leq i \leq k$, then it switches to q', writes b_i onto the i-th tape and moves its i-th head according to M_i, for all i with $1 \leq i \leq k$.

Unless stated otherwise, assume that the input to a k-tape TM is on the first tape, and that all other tapes are filled with blanks. However, sometimes it is more appropriate to consider k-tape TMs that have an additional *input tape* (thus $k + 1$ tapes in total) which contains the input and cannot be modified (i.e., the new symbol written there is always identical to the old symbol).

We will now see that the computational power of multi-tape TMs is the same as that of single-tape TMs. This illustrates the robustness of TMs as a model of computation.

To state the result precisely, we introduce measures for the time and space consumption of TMs. For a TM M, we use

- $\mathsf{time}_M(w)$ to denote the length of the computation of M on input w;
- $\mathsf{space}_M(w)$ to denote the maximum number of cells used by M on any tape when started on input w; if the Turing machine is equipped with an input tape, the cells consumed on this tape are not counted.

Note that both $\mathsf{time}_M(w)$ and $\mathsf{space}_M(w)$ can be ∞. The reason for not counting the space on the input tape is that there are interesting cases where a TM needs less space than occupied by the input, see Exercise 2.

Let $T, S : \mathbb{N} \to \mathbb{N}$ be functions. Then a TM M is called

- *T-time bounded* if $\mathsf{time}_M(w) \leq T(|w|)$ for all $w \in \Sigma^*$;
- *S-space bounded* if $\mathsf{space}_M(w) \leq S(|w|)$ for all $w \in \Sigma^*$,

where $|w|$ denotes the length of w. Note that if a TM is T-time bounded for any T, then it stops on every input. Note that $S(n) \geq n$ unless there is an extra input tape. Also note that time bounds T with $T < T(n)$ do not make much sense because within such time bound not even the input can be read. We always assume that time bounds T satisfy $T(n) \geq n$ for all $n \in \mathbb{N}$.

Theorem 2. *Given a T-time bounded and S-space bounded k-tape TM M, we can construct a $2T^2$-time bounded and S-space bounded (1-tape) TM M' such that $L(M) = L(M')$.*

Proof. (sketch). Let $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$. Central ideas for representing k tapes in a single one:

- use alphabet Γ^k, i.e., tuples $(a_1, \dots, a_k)$ with $a_i \in \Gamma$ for $1 \leq i \leq k$;
- the difference between a 1-tape TM M operating on Γ^k and a k-tape TM M' operating on Γ is that M' can move its heads independently. To simulate this, we use the alphabet $\widehat{\Gamma} := (\Gamma \cup \{a^* \mid a \in \Gamma\})^k$:

a	b	b	b^*	a	b	b	a	
a	a	a	a	b	a	a	a^*	$\cdots$
b	a^*	b	b	a	a	b	a	

- since M' should have input alphabet Σ, M' uses alphabet $\Sigma \cup \widehat{\Gamma}$.

The TM M' simulates the computation of M:

1. M' first replaces every input symbol $a \in \Sigma$ with $(a, \sqcup, \dots, \sqcup)$;
2. for each step of M, M' traverses the tape left-to-right and collects all symbols under the heads; then M' determines the move of M and traverses back right-to-left, making all necessary changes;
3. M' accepts/rejects whenever the simulated M reaches the accepting/rejecting state.

Step 1 can be done during the first traversal. We have:

- M' is $2T^2$-time bounded: for every step of M, M' needs to traverse the tape twice. The maximum length of the tape is $T(n)$ since a TM can write at most a single symbol in every step. Therefore, simulating a single step of M requires $2T(n)$ steps of M'.
- M' is S-space bounded: obviously, M' does not use more space than M.

❏

In complexity theory, we usually abstract from concrete constants such as the factor two in Theorem 2. It is thus convenient to use Landau notation, in particular "big $\mathcal{O}$": for f, g functions from $\mathbb{N}$ to $\mathbb{N}$, we write

- $f(n) \in \mathcal{O}(g(n))$ if $\exists c \in \mathbb{N}^+ : \exists n_0 \in \mathbb{N} : \forall n > n_0 : f(n) \leq c \cdot g(n)$

which intuitively means that f grows as g or slower. An alternative definition is $f(n) \in \mathcal{O}(g(n))$ if $\lim \frac{f(n)}{g(n)} \leq c$. Sometimes, it will be necessary to say that a function f grows strictly slower than a function g. We use "small o":

- $f(n) \in o(g(n))$ if $\forall c \in \mathbb{N}^+ : \exists n_0 \in \mathbb{N} : \forall n > n_0 : c \cdot f(n) \leq g(n)$

The alternative definition is $f(n) \in o(g(n))$ if $\lim \frac{f(n)}{g(n)} = 0$. Some examples are given in Exercise 4.

2.3 Nondeterministic Turing Machines

A (single-tape) non-deterministic TM (NTM) is much like a deterministic one (DTM). The main difference is that the machine is not restricted to having a single, uniquely defined next action. Instead, it has the *choice* between several actions. To formalize this, we use a transition function

$$\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Thus, for each state-symbol combination (q, a), there is a *set* $\delta(q, a)$ of possible next steps. Equivalently to this generalized transition function, we will use a transition *relation*

$$\Delta \subseteq (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \times Q \times \Gamma \times \{L, R\}.$$

The $\vdash_M$ and $\vdash^*$ relations are defined in an analogous way as for DTMs, only that $\vdash_M$ is no longer a total function, i.e., for a configuration c, there can be more (and less) than one configuration c' such that $c \vdash_M c'$.

On a single input w, an NTM may thus have many different computations. Some of them may be accepting, some rejecting, and some may not terminate. We say that an NTM M *accepts* an input w iff there *exists* an accepting computation of M on w. Note the asymmetry: it follows that if an input w is *not* accepted, then *all* computations starting on w are rejecting or infinite.

The language $L(M)$ accepted by an NTM M is defined as for DTMs. We say that an NTM M stops on input w if *all* computations of M on w are finite. We say that M *decides* a language L if $L(M) = L$ and M stops on all inputs.

The functions $\mathsf{time}_M(w)$ and $\mathsf{space}_M(w)$ are defined as for DTMs, but referring to the *maxium* length/space consumption of *all* computations of M when started on w. Finally, being T-*time bounded* and S-*space bounded* is defined exactly as for DTMs.

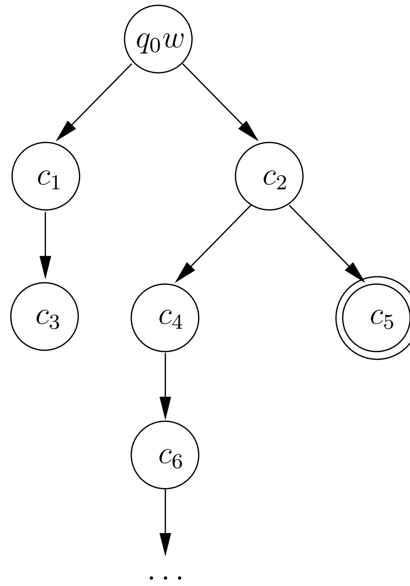
Note that the set $\delta(q, a)$ can also be empty, which means that the TM cannot make any further steps when being in a configuration $c = uqav$; we call c a *blocking*

configuration and say that the TM *blocks* if it reaches such a configuration. Intuitively, a blocking configuration $uqav$ with $q \neq q_{\mathsf{acc}}$ is very similar to a rejecting configuration.

Although NTMs are a considerably less realistic model of computation than DTMs, they play a very important role in complexity theory because they capture in a very natural way a large group of problems occurring in computer science.

2.4 DTM vs. NTM

We want to show that DTMs and NTMs decide the same classes of languages. One direction is trivial: every DTM is also an NTM. For the other direction, it is common to arrange all the computations of an NTM on a given input w in a tree:



This view suggests a way to convert an NTM into a DTM that accepts the same language (using breadth-first search).

Theorem 3. *For every T-time bounded and S-space bounded NTM M, there is a $2^{\mathcal{O}(T(n))}$-time bounded and $\mathcal{O}(T)$-space bounded DTM M' such that $L(M) = L(M')$.*

Proof. (sketch) Let $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$. Define a set of choices and a corresponding alphabet

$$\begin{aligned} C &:= \{(q, a, M) \mid \exists q', a' : (q', a', q, a, M) \in \Delta\} \\ \Sigma_{ch} &:= \{a_{q,b,M} \mid (q, b, M) \in C\}. \end{aligned}$$

Set $d := |C|$. The DTM M' simulates M using three tapes. Basic idea:

- the first tape of M' contains the input and is not modified;
- every computation of M can be viewed as a finite sequence of choices, with each choice determining the next configuration (but there are sequences of choices that do not describe a computation);

- M' successively generates all sequences of symbols from Σ_{ch} on the second tape in order of increasing length;
- for each generated sequence, M' simulates the computation of M corresponding to the sequence on the third tape (if it exists), keeping at most one configuration of M at a time on the tape;
- M' accepts if it encounters the accepting configuration of M during simulation;
- M' rejects if it encounters a $t \in \mathbb{N}$ such that there is no computation of length t.

The number of nodes in the computation tree is bounded by $d^{T(n)+1} - 1$ and it can be seen that each computation can be simulated using at most $c \cdot T(n)$ steps for some constant c. Now, $(d^{T(n)+1} - 1) \cdot c \cdot T(n) \in 2^{\mathcal{O}(T(n))}$ (Exercise). Since all computations of M are of length at most $T(n)$, the number of cells used on tapes 2 and 3 is bounded by $\mathcal{O}(T(n))$. ❏

Note that the above construction works without knowledge of $T(M)$, and thus it actually provides an algorithm for converting a given NTM into a DTM.

3 Complexity Classes

A complexity class is determined by a machine model (such as DTM or NTM) and resource bounds for that machine model, with typical resources being time and space. Then, the associated complexity class is the class of problems (i.e., formal languages) that can be decided by a machine of the given model within the given bounds.

Definition 4. Let T and S be functions from $\mathbb{N}$ to $\mathbb{N}$. Then

- $\mathsf{DTime}_k(T)$ is the class of all languages L that are decided by a T-time bounded k-tape DTM;
- $\mathsf{DTime}(T) := \bigcup_{k \geq 1} \mathsf{DTime}_k(T)$;
- $\mathsf{DSpace}_k(S)$ is the class of all languages L that are decided by a S-space bounded k-tape DTM;
- $\mathsf{DSpace}(S) := \bigcup_{k \geq 1} \mathsf{DSpace}_k(S)$;
- $\mathsf{NTime}_k(T)$, $\mathsf{NTime}(T)$, $\mathsf{NSpace}_k(S)$, and $\mathsf{NSpace}(S)$ are defined analogously based on NTMs.

△

Note that what we are defining here are classes of *worst case* complexity: by definition of T-time boundedness, we require that $T(n)$ is an upper bound for the time consumption on *all* words of length n which may include cases that are fairly complex and "untypical" for the problem (and similarly for space boundedness). An alternative is

to study *average case complexity*, but this requires a mathematical characterization of what an average case actually is, and this is often difficult to attain.

Some frequently used complexity classes have special names, most importantly the following:

- P $:= \bigcup_{d\in\mathbb{N}} \mathsf{DTime}(n^d)$ and NP $:= \bigcup_{d\in\mathbb{N}} \mathsf{NTime}(n^d)$;
- PSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{DSpace}(n^d)$ and NPSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{NSpace}(n^d)$;
- ExpTime $:= \bigcup_{d\in\mathbb{N}} \mathsf{DTime}(2^{n^d})$ and NExpTime $:= \bigcup_{d\in\mathbb{N}} \mathsf{NTime}(2^{n^d})$;
- ExpSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{DSpace}(2^{n^d})$ and NExpSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{NSpace}(2^{n^d})$;
- LogSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{DSpace}(d \cdot \log n)$ and NLogSpace $:= \bigcup_{d\in\mathbb{N}} \mathsf{NSpace}(d \cdot \log n)$.

In the definition of LogSpace and NLogSpace, we assume TMs to be equipped with an additional input tape. Observe that a LogTime complexity class does not make much sense because then $T(n) < n$.

The main aim of *studying* complexity theory is to understand how complexity classes are interrelated. The main aim of *applying* complexity theory is to determine the position that a given problem has in the landscape of complexity classes.

Here are some basic observations regarding the relationship between time and space complexity classes.

Lemma 5.

1. $\mathsf{DTime}(T) \subseteq \mathsf{DSpace}(T)$ *and* $\mathsf{NTime}(T) \subseteq \mathsf{NSpace}(T)$*;*
2. $\mathsf{DSpace}(S) \subseteq \mathsf{DTime}(2^{\mathcal{O}(S)})$ *and* $\mathsf{NSpace}(S) \subseteq \mathsf{NTime}(2^{\mathcal{O}(S)})$*;*

Proof. Let M be a k-tape NTM (in particular, M could also be a DTM).

1. If M is T-time bounded, then it is also T-space bounded because M can only use a single tape cell (on each tape) in every step that it makes.
2. Follows from the following:
   - the number of configurations of M on an input of length n is bounded by the following number, where $m = S(n)$:

     $$|Q| \cdot m^k \cdot |\Gamma|^{k \cdot m} \leq 2^{\mathcal{O}(m)}$$

   - in every computation on some input w, each configuration appears at most once since otherwise M has at least one infinite computation on w.

❏

Regarding the relationship of deterministic and non-deterministic complexity classes, we trivially have the following.

Lemma 6. $\mathsf{DTime}(T) \subseteq \mathsf{NTime}(T)$ *and* $\mathsf{DSpace}(S) \subseteq \mathsf{NSpace}(S)$.

We can easily derive the following inclusions:

$$\begin{array}{ccccccccc}
\textsc{LogSpace} & \subseteq_{(2)} & \mathrm{P} & \subseteq_{(1)} & \textsc{PSpace} & \subseteq_{(2)} & \textsc{ExpTime} & \subseteq_{(1)} & \textsc{ExpSpace} \\
|\cap_{(6)} & & |\cap_{(6)} & & |\cap_{(6)} & & |\cap_{(6)} & & |\cap_{(6)} \\
\textsc{NLogSpace} & \subseteq_{(2)} & \mathrm{NP} & \subseteq_{(1)} & \textsc{NPSpace} & \subseteq_{(2)} & \textsc{NExpTime} & \subseteq_{(1)} & \textsc{NExpSpace}
\end{array}$$

There is another easy connection between complexity classes. To state it, we restrict ourselves to "well-behaved" time and space bounds.

Definition 7. A function $T : \mathbb{N} \to \mathbb{N}$ is *time constructible* if there is a DTM M with $\mathsf{time}_M(w) = T(|w|)$ for all w. Analogously, $S : \mathbb{N} \to \mathbb{N}$ is *space constructible* if there is a DTM M with $\mathsf{space}_M(w) = S(|w|)$ for all w. $\triangle$

Standard functions such as n^d, 2^n, and $n!$ are time and space constructible. Also, these function classes are closed under operations such as $f + g$, $f \cdot g$, 2^f, and f^g.

Lemma 8. *Let S be a space constructible function. Then* $\mathsf{NSpace}(S) \subseteq \mathsf{DTime}(2^{\mathcal{O}(S)})$.

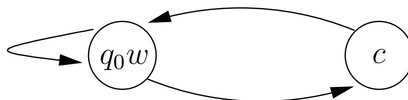
To prove Lemma 8, we need some preliminaries.

Definition 9. Let M be a (1-tape) NTM and $s \in \mathbb{N}$. Then $\mathsf{Conf}_M(s)$ is the set of all configurations of M in which at most s tape cells are labelled with a non-blank symbol. The *s-configuration graph* for M is the directed graph $G_M(s) = (V, E)$, where

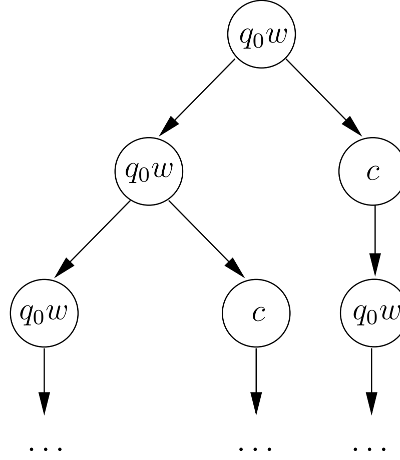
- $V = \mathsf{Conf}_M(s)$;
- $E = \{(c, c') \mid c \vdash_M c'\}$.

$\triangle$

There is an obvious relation between configuration graphs and configuration trees as used in Section 2.4. Let M be an S-space bounded NTM, w an input of length n, and $G_M(S(w)) = (V, E)$. Then the configuration tree for M on w is obtained by using the initial configuration $q_0 w \in V$ as the root, and "unravelling" $G_M(S(n))$. For example, unravelling



yields



The following is obvious.

Lemma 10. *Let M be an S-space bounded NTM and $w \in \Sigma^*$ of length n. Then $w \in L(M)$ if in $G_M(S(n))$, an accepting computation is reachable from q_0w.*

We are now ready to prove Lemma 8.

Proof. If $L \in \mathsf{NSpace}(S)$, then by Theorem 2 there is an S-space bounded 1-tape NTM M with $L(M) = L$. We construct a DTM M' that does the following on input w of length n:

- determine $S(n)$ by running the TM M_S witnessing space constructability of S on a separate tape. The number of tape cells used encodes $S(n)$ in unary.
- construct the graph $G_M(S(n))$, i.e.
  - generate all configurations: words from $(Q \cup \Gamma)^*$ of length at most $S(n)+1$ with exactly one symbol from Q;
  - for each pair of configurations c and c', compute whether $c \vdash c'$.
- for every generated accepting configuration c, check whether there is a path from the initial configuration to c.

M_S runs in time at most $2^{O(n)}$ (see proof of Lemma 5); there are at most $2^{O(n)}$ configurations in $G_M(S(n))$, and thus (i) $G_M(S(n))$ is generated in time at most $2^{O(n)}$ and (ii) at most $2^{O(n)}$ reachability problems need to be solved. A single such problem can be solved in polynomial time (as we will see later). ❏

Observe that the construction is not effective since we need to know M_S.

This construction also works when we consider S-space bounded TMs equipped with an extra input tape and $S(n) \geq \log(n)$ (without an extra input tape, $S(n) \geq n$ by definition). Then, the input tape content is not part of the configuration, but the head position on the input tape is. There are at most $n = 2^{\log n}$ such positions. Thus we get:

Corollary 11. NLogSpace $\subseteq$ P *and* NPSpace $\subseteq$ ExpTime.

The updated picture is as follows:

$$\textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{P} \subseteq \frac{\textsc{NP}}{\textsc{PSpace}} \subseteq \textsc{NPSpace} \subseteq \textsc{ExpTime} \subseteq \frac{\textsc{NExpTime}}{\textsc{ExpSpace}} \subseteq \textsc{NExpSpace}.$$

It remains to clean up the messy parts. To prove the following theorem, we again view acceptance of Turing machines as reachability in the configuration graph, but exploit this in a much smarter way.

Theorem 12 (Savitch). *If S is space constructible, then* $\mathsf{NSpace}(S) \subseteq \mathsf{DSpace}(S^2)$.

Proof. Let M be an S-space bounded NTM. We convert M into a 3-tape DTM M' as follows. Let w be an input to M' of length n. Clearly, M' cannot generate the whole graph $G_M(S(n)) = (V, E)$ since it has $2^{\mathcal{O}(S(n))}$ nodes. Main ideas:

- let $\mathsf{Path}(c, c', i)$ be true if there is a path of length at most 2^i from c to c' in $G_M(S(n))$. Also, let $N = |V|$. Clearly, M accepts w iff $\mathsf{Path}(q_0w, c_{\mathsf{acc}}, \log N)$ is true for some accepting configuration c_{acc};
- any path of length at most 2^i from c to c' has a midpoint m, and the subpaths from c to m and from m to c' are of length at most 2^{i-1};
- use divide and conquer: to decide $\mathsf{Path}(c, c', i)$, consider all configurations m as potential midpoints and recursively check $\mathsf{Path}(c, m, i-1)$ and $\mathsf{Path}(m, c', i-1)$.

On tape 1, M' first generates $S(n)$ and then uses it to compute N (in binary). Then, on Tape 2 we successively generate all accepting configurations c_{acc} in $G_M(S(n))$, keeping only one at a time on the tape. For each c_{acc}, it runs the following algorithm accepting if the algorithm succeeds on at least one c_{acc}:

1. first write the triple $(q_0w, c_{\mathsf{acc}}, \log N)$ on tape 3;
2. if the right-most triple on tape 3 is $(c, c', 0)$, then return a positive answer if $c = c'$ or $c \vdash_M c'$;
3. consider the right-most triple (c, c', i) on tape 3, generate first midpoint candidate m, append triple $(c, m, i-1)$ to tape 3, recursively start with 2.;
4. if the answer was negative, erase triple $(c, m, i-1)$ and try next m; if it is positive, erase $(c, m, i-1)$, write $(m, c', i-1)$ (by recovering c' from the adjacent triple to the left) and recursively start with 2.;
5. if the answer is negative, erase $(m, c', i-1)$ and try next m; if it is positive, answer positively.

On Tapes 1 and 2, we need at most $\mathcal{O}(S(n))$ cells. Tape 3 is used like a stack in recursive procedure calls. It contains at most $\log N \in \mathcal{O}(S(n))$ triples since we start with $i = \log N$. Each triple is of length at most $\mathcal{O}(S(n))$, thus $\mathcal{O}(S(n)^2)$ tape cells are consumed. ❑

Corollary 13. PSPACE = NPSPACE *and* EXPSPACE = NEXPSPACE.

Observe that Savitch's Theorem does not apply to LOGSPACE and NLOGSPACE because of the quadratic blowup. However, it is the consequence of a theorem by Immerman and Szelepcsényi, that LOGSPACE = NLOGSPACE.

We finally obtain the following picture, in which NLOGSPACE, NPSPACE, and NEXPSPACE do not occur anymore:

$$\text{LogSpace} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{ExpSpace}.$$

An important question is which of these inclusions is strict. One of the major frustrations (and fascinations!) of complexity theory is that regarding this question, there are much more open problems than settled ones.

For example, note that we have not proved a version of Savitch's theorem for time complexity classes. Such a theorem would e.g. imply P = NP. It is not known whether this equality holds. Discussing this issue is the subject of the next section.

4 Tractable versus Intractable

The most important distinction in computational complexity is between problems that can be solved "efficiently" and those that cannot. Problems of the former kind are commonly called *tractable*. Let us consider two example of problems on graphs: one of them tractable and the other not.

4.1 Examples

Recall: the graph reachability problem is to decide, given a graph $G = (V, E)$ and two nodes $v, v' \in V$, whether there are nodes $v_0, \ldots, v_n$ $(n \geq 0)$ such that $v = v_0$, $v' = v_n$, and $(v_i, v_{i+1}) \in E$ for all $i < n$. We write this as

$$\text{REACH} := \{(G, v, v') \mid G = (V, E) \text{ is such that } v' \text{ is reachable from } v\}$$

because we view a problem as a formal language (which is a set). We neglect encoding issues in this presentation.

Consider the following algorithm, given in pseudocode (for a TM implementation, we need a suitable encoding of graphs as words):

```
S := {v}
mark v
while S ≠ ∅ do
   choose u ∈ S
   S := S \ {u}
   for each edge (u, u') ∈ E do
      if u' is not marked then
         mark u'
         S := S ∪ {u'}
```

```
        endif
    endfor
endwhile
if v' is marked then answer "yes", else answer "no".
```

This algorithm is correct and terminates after at most $\mathcal{O}(n^2)$ steps, where n is the number of nodes:

- each node appears at most once in S, and thus the while loop makes at most n steps;
- since each node directly reaches at most n nodes, the for loop is executed at most n times in each iteration of the while loop.

(Actually, it can be seen to terminate after $\mathcal{O}(n+m)$ steps where m is the number of edges). A runtime behaviour of $\mathcal{O}(n^2)$ can be considered relatively efficient. E.g., on an input of size $n = 1000$, the algorithm needs $1000^2 \leq 2^{20}$ steps. Compare this to the $\approx 2^{30}$ ($= 1024 \cdot 2^{20}$) steps per second made by a 1Ghz computer.

An *undirected graph* is a pair $G = (V, E)$ with V a set of nodes and E a set of sets $\{u, v\} \subseteq V$ of cardinality two. A k-*clique* in G is a set $C \subseteq V$ of cardinality k such that $\{u, v\} \in E$ for all $u, v \in C$ with $u \neq v$. The *clique problem* is

$$\text{CLIQUE} := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

It is not important whether k is coded in unary or binary since we may assume that k is at most $|V|$.

The following algorithm for CLIQUE springs to mind:

```
for all S ⊆ V of size k
    f := 1
    for all u, v ∈ S with u ≠ v
        if (u, v) ∉ E then f := 0
    endfor
    if f = 1 then return "yes"
endfor
return "no"
```

This algorithm is much less efficient:

- the for loop makes $\binom{n}{k}$ steps;
- the inner loop makes k^2 steps.

Take e.g. $k = n/2$. Then $\binom{n}{k} \geq 2^n$, i.e., the algorithm has exponential runtime behaviour. If $n = 1000$ and $k = 500$, the algorithm thus needs more than 2^{1000} steps. A 1Ghz computer will thus need 2^{970} seconds (a number with 276 digits). A natural

question is whether all algorithms deciding CLIQUE *have to be* that hard. After all, there could be a more clever algorithm that runs in polynomial time.

These two examples suggest that a problem is tractable if it can be solved by a DTM with polynomial time consumption. Indeed, this is the classical view of complexity theory: a problem is tractable if it is in P. Note:

- Runtimes such as n^{1000} are polynomial, but probably not tractable; $2^{\frac{n}{1000}}$ is exponential, but probably tractable. In practice, such extreme rates of growth rarely appear.

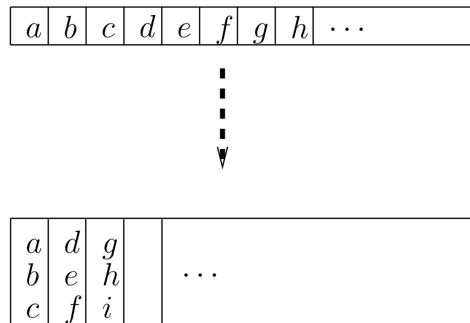
- The tractable/non-tractable distinction is only a rule of thumb and there are exceptions. For example, in linear programming the polynomial ellipsoid algorithm is unusable whereas the worst-case exponential simplex method has a superb perfomance in practice.

4.2 The Speedup Theorem

To understand tractability, we start with the fundamental question whether allowing more resources actually allows us to solve more problems. We first show that adding multiplicative constants does usually not allow us to solve additional problems, i.e., it is possible to speed up most DTMs by any constant factor.

Theorem 14 (Speedup Theorem). *Let* $T : \mathbb{N} \to \mathbb{N}$ *be such that* $n \in o(T)$*. Then* $L \in \mathsf{DTime}(T)$ *implies* $L \in \mathsf{DTime}(\max\{n, \lceil \epsilon \cdot T \rceil\})$ *for all* $\epsilon \in (0, 1]$*.*

Proof. Let M be a k-tape DTM deciding L in time $T(n)$. We construct a $k+1$-tape TM M' that works over alphabet $\Sigma \cup \Sigma^m$ for some m that we fix later. Thus, M' can represent $m > 1$ symbols of M in a single symbol. E.g. if $m = 3$:



This saves space, but also allows to execute multiple steps in one. M' works as follows:

- M' copies the input of M onto the new tape, compressing m symbols of Σ into one of Σ^m; this and bringing back the head on the new tape needs $n + \lceil \frac{n}{m} \rceil$ steps;

- M' simulates m steps of M using eight steps. To do this, the following steps are perfomed simultaneously on each tape:

- M' saves "in its state set" the content of the neighboring cells of the current cell. This is possible since it is only a constant amount ($2m$ symbols) of information. It requires the four steps left,right,right,left.
- In m steps, M can only access tape cells which are at distance at most m from the current cell. In M', all these cells are represented in the current cell and its direct neighbors. Thus, M' has enough information to compute the next m steps of M (this is hard-wired into M'). To execute the changes made in these steps, M' again needs four steps.

- regarding acceptance/rejection, M' behaves like M.

On inputs of length n, M' makes

$$n + \lceil \frac{n}{m} \rceil + 8\lceil \frac{T(n)}{m} \rceil \leq n + \frac{n}{m} + 8\frac{T(n)}{m} + 2$$

steps. Since $n \in o(T)$, there is an n_0 such that for all $n > n_0$, $n \leq \frac{T(n)}{m}$. Assume w.l.o.g. that $n_0 \geq 1$. On inputs of length $n \geq n_0$, M' makes the following number of steps:

$$\begin{aligned} n + \tfrac{n}{m} + 8\tfrac{T(n)}{m} + 2 &\leq 2n + \tfrac{n}{m} + 8\tfrac{T(n)}{m} && (n > n_0 \geq 1 \text{ implies } 2n \geq n+2) \\ &\leq 2\tfrac{T(n)}{m} + \tfrac{T(n)}{m^2} + 8\tfrac{T(n)}{m} && (n > n_0 \text{ implies } n \leq \tfrac{T(n)}{m}) \\ &\leq T(n) \cdot \tfrac{11}{m} \end{aligned}$$

Now we fix m such that $\varepsilon \cdot m \geq 11$. Then $T(n) \cdot \frac{11}{m} \leq \varepsilon \cdot T(n)$ and thus the machine is required on all inputs of length greater than n_0. There are only finitely many remaining inputs (those of length $n \leq n_0$), which can be coded into the transition function: once the input has been read completely, the machine immediately accepts or rejects. Clearly, the runtime is within $\max\{n, \lceil \epsilon \cdot T \rceil\}$. ❏

Theorem 14 shows that if a TM operates in superlinear time $c \cdot f(n)$, then c can be made arbitrarily small. This justifies our use of the $\mathcal{O}$ notation. Theorem 14 also justifies our definition of P in terms of polynomials of the form n^d. Consider e.g. the polynomial $14n^2 + 31n$. It is bounded by $45n^2$ and in the latter we can make the coefficient equal to one by Theorem 14.

There is an analogous theorem that refers to space consumption rather than to time consumption.

Theorem 15 (Space Compression Theorem). *Let* $S : \mathbb{N} \to \mathbb{N}$. *Then* $L \in \mathsf{DSpace}(S)$ *implies* $L \in \mathsf{DSpace}(\max\{n, \lceil \epsilon \cdot S \rceil\})$ *for all* $\epsilon \in (0, 1]$.

There are a bunch of related theorems, showing e.g. that also linear time machines can be sped up. A speedup theorem for non-deterministic machines exists as well.

4.3 The Hierarchy Theorem

We show that if we are more generous when increasing resources, then we can indeed solve more problems. Results showing this are known as hierarchy theorems. We start with a special case.

Theorem 16. $\mathrm{P} \subsetneq \textsc{ExpTime}$.

We thus have the following situation:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{P}\textsc{Space} \subseteq \textsc{ExpTime}$$

$$\subsetneq$$

It follows that one of the three upper inclusions is strict, but it is unknown which one this is.

To prove Theorem 16, we represent TMs as words over some fixed alphabet Σ_{TM} and thus can use them as input to TMs.

W.l.o.g., we assume that the set of states of a TM is of the form $\{1, \ldots, k\}$ for some $k \in \mathbb{N}$, where 1 is q_0, 2 is q_{acc}, and 3 is q_{rej}. The set of symbols is of the same form. We can represent states and symbols by their binary encoding over the alphabet $\{0, 1\} \subseteq \Sigma_{TM}$. Encoding the symbols is necessary since TMs may use alphabets different from Σ_{TM}.

Representing the TM boils down to representing the transition relation, from which the number of tapes, the set of states and the set of symbols can be recovered. We assume a fixed representation scheme without giving details. $\widehat{M}$ denotes the Σ_{TM}-encoding of the TM M and $\widehat{w}$ the Σ_{TM}-encoding of the input word w.

Consider the following language over $\Sigma_{TM} \uplus \{;\}$:

$$H := \{\widehat{M}; \widehat{w} \mid \text{DTM } M \text{ accepts } w \text{ after at most } 2^{|w|} \text{ steps}\}.$$

Lemma 17. $H \notin \mathrm{P}$.

Proof. Assume $H \in \mathrm{P}$. By Lemma 2, there is a (1-tape) DTM M_H such that $L(M) = H$ and M_H is n^d-time bounded for some $d \in \mathbb{N}$. Assume w.l.o.g. that $d \geq 2$. Define a new DTM D:

- D duplicates its input w to $w; w$ and then behaves like M_H. It rejects if M_H accepts and vice versa.

Since $d \geq 2$ and duplication takes $\mathcal{O}(n^2)$ time, D is $f_D(n)$-time bounded, where $f_D(n) \in \mathcal{O}((2n)^d)$. Let $m := |\widehat{D}|$. "Blow up" $\widehat{D}$ by introducing useless transitions so that $2^m > f_D(m)$. Does D accept $\widehat{D}$? There are two possibilities:

- Yes. Then M_H rejects $\widehat{D}; \widehat{D}$, which implies $\widehat{D}; \widehat{D} \notin H$. By definition of H, D does not accept $\widehat{D}$ within 2^m steps. Since $2^m > f_D(m)$ and D is f_D-time bounded, D rejects $\widehat{D}$. Contradiction.

- No. Then M_H accepts $\widehat{D}; \widehat{D}$, implying $\widehat{D}; \widehat{D} \in H$ and thus D accepts $\widehat{D}$. Contradiction.

❏

To prove Theorem 16, it remains to show the following

Lemma 18. *$H \in$ ExpTime.*

Proof. Define a TM M' that behaves as follows:

1. check if the input is of the form $\widehat{M};\widehat{w}$ with M DTM, reject if not;
2. save a copy of $\widehat{M}$ on a separate tape;
3. simulate M on w by "interpreting" $\widehat{M}$. During the simulation, Tape 1 contains the encoded contents of the tape of M.

   This can be done such that simulating each step of M needs $\mathcal{O}(|\widehat{M}|)$ steps;
4. use an additional tape to simultaneously count in binary from 0 to $2^{|w|}$, incrementing for each step of M;
5. accept if M accepts w and reject if M rejects or did not terminate after $2^{|w|}$ steps.

By construction, $L(M) = D$ and $\mathsf{time}_D(w) \in \mathcal{O}(n \cdot 2^n)$. ❑

What we have just seen is a special case of a much more general theorem called the *hierarchy theorem*, whose proof is exactly analogous to the proof of this special case.

Theorem 19 (Time Hierarchy Theorem). *Let $T, t : \mathbb{N} \to \mathbb{N}$ such that $T(n) \geq n$, T time constructible, and $t \cdot \log t \in o(T)$ (i.e., $t \cdot \log t$ grows slower than T). Then $\mathsf{DTime}(t) \subsetneq \mathsf{DTime}(T)$.*

It is important that T is time constructible: in the proof of Lemma 18, we needed to count 2^n steps while simulating a TM. In the general case, we need to count $T(n)$ steps and this is achieved by simulating the TM witnessing time constructibility of T.

Theorem 19 implies Theorem 16: since every polynomial p ultimately becomes smaller than 2^n, P is a subset of $\mathsf{DTime}(2^n)$. By Theorem 19, $\mathsf{DTime}(2^n) \subsetneq \mathsf{DTime}(2^{n^2})$, the latter being a subset of ExpTime.

Theorem 19 also yields lot of additional results such as $\mathsf{DTime}(n^d) \subsetneq \mathsf{DTime}(n^{d+1})$ for all $d \in \mathbb{N}$ ($\log(n^d) \in o(n)$, and thus $n^d \cdot \log(n^d) \in o(n^{d+1})$).

There is also an analogous non-deterministic version of the time hierarchy theorem. For example, it yields NP $\subsetneq$ NExpTime. Finally, there is a space hierarchy theorem. Its proof is similar to the proof of the time hierarchy theorem.

Theorem 20 (Space Hierarchy Theorem). *Let $S, s : \mathbb{N} \to \mathbb{N}$ such that $S(n) \geq \log n$, S space constructible, and $s \in o(S)$. Then $\mathsf{DSpace}(s) \subsetneq \mathsf{DSpace}(S)$.*

This theorem yields, e.g., PSpace $\subsetneq$ ExpSpace and $\mathsf{DSpace}(n^d) \subsetneq \mathsf{DSpace}(n^{d+1})$ for all $d \in \mathbb{N}$.

4.4 Basics of NP

Let us return to the question whether there is a better algorithm for CLIQUE. Is CLIQUE one of the problems in ExpTime $\setminus$ P? Unfortunately, nobody knows, i.e., all known (deterministic) algorithms require exponential time, but nobody was able to prove that this is *necessarily* the case. This is closely related to the famous "P $\neq$ NP" problem.

Recall that P $\subseteq$ NP $\subseteq$ ExpTime. The following simple non-deterministic algorithm shows that CLIQUE is in NP:

guess a subset $S \subseteq V$ of size k
for all $v, v' \in S$ with $v \neq v'$
  if $(v, v') \notin E$ then return "no"
endfor
return "yes"

Here, "guessing" a subset $S \subseteq V$ of size k means that the Turing machine non-deterministically chooses k elements of V, avoiding repetitions. For example, an NTM can choose a value from $\{0, 1\}$ by the transitions

$$(q, a, q', 0, R) \text{ and } (q, a, q', 1, R).$$

Since graphs and also nodes of a graph are encoded as words, choosing a node actually involves a (polynomial) number of such choices. Recall that an NTM accepts its input if there *exists* an accepting configuration that is reachable from the initial configuration. Intuitively, this means that an NTM always makes the correct guess if possible at all.

It is easily seen that the above algorithm needs only time $\mathcal{O}(n^2)$ steps and thus witnesses membership in NP.

The principle underlying the above algorithm can be generalized to an alternative definition of the complexity class NP.

Definition 21. Let $L \subseteq \Sigma^*$. A relation $R \subseteq \Sigma^* \times \Gamma^*$ is a *proof relation* for L if

- Soundness. $(w, p) \in R$ implies $w \in L$.
- Completeness. $w \in L$ implies $(w, p) \in R$ for some $p \in \Gamma^*$.

If $(w, p) \in R$, then p is called a *proof* for w being in L. R is *polytime verifiable* if

1. there is a $k \geq 1$ such that $|p| \leq |w|^k$ for all $(w, p) \in R$;
2. it is in P to decide, given a $(w, p) \in \Sigma^* \times \Gamma^*$, whether $(w, p) \in R$.

△

As illustrated by the above algorithm, CLIQUE has a polytime-verifiable proof relation:

$$\{((G,k),S) \mid S \subseteq V \text{ is a } k\text{-clique in } G\}.$$

Other problems do probably not have such proof relations, consider e.g. the problem to decide, given a 2^n-time bounded DTM M and a word w, whether M accepts w.

Theorem 22. *For all $L \subseteq \Sigma^*$, $L \in \text{NP}$ iff L is polytime verifiable.*

Proof. "$\Leftarrow$". If R is a polytime-verifiable proof relation for L with bound $|w|^k$ on the proof length, then a polynomially time bounded NTM can solve L by guessing a $p \in \Gamma^*$ of length at most $|w|^k$ and then checking in polytime whether $(w,p) \in R$.

"$\Rightarrow$". If $L \in \text{NP}$, there is an NTM M that decides L in time n^k, for some k. Define R as follows: $(w,p) \in R$ iff p is the encoding of an accepting computation of M on input w. Clearly, R is a proof relation for L. Since M runs in polytime and it is possible to check in polytime whether a sequence of configurations is an accepting computation of an NTM, R is polytime-verifiable. ❑

Thus, a problem is in NP iff (i) every "yes" instance has a succint proof of it being a "yes" instance and (ii) proofs can be checked efficiently. Note that this definition does not at all use non-deterministic TMs.

We have seen that CLIQUE is in NP. Since $\text{P} \subseteq \text{NP}$, however, this doesn't tell us much about whether or not CLIQUE is in P. More insight can be gained through the notion of *completeness*. Intuitively, a problem is NP-complete if it belongs to the hardest problems in NP.

Definition 23. Let $L \subseteq \Sigma^*$ and $L' \subseteq \Gamma^*$. A *polytime reduction* from L to L' is a function $f : \Sigma^* \to \Gamma^*$ such that

- $w \in L$ gdw. $f(w) \in L'$ for all $w \in \Sigma^*$;
- f can be computed in polynomial time.

If there is such a reduction, we say that L is *polynomially reducible* to L' and write $L \leq_p L'$. △

Intuitively, $L \leq_p L'$ if L' is at least as hard as L. This underlies the following lemma.

Lemma 24. *Let $\mathcal{C} \in \{\text{P}, \text{NP}, \text{PSpace}, \text{ExpTime}, \text{NExpTime}, \text{ExpSpace}\}$. Then $\mathcal{C}$ is* closed under polytime reductions*, i.e., if $L' \in \mathcal{C}$ and $L \leq_p L'$, then $L \in \mathcal{C}$.*

For LogSpace, this is not true. The appropriate reduction for those classes is a *logspace reduction*, defined like a polytime reduction but computable in logarithmic space.

Definition 25. Let $\mathcal{C} \in \{\text{P}, \text{NP}, \text{PSpace}, \text{ExpTime}, \text{NExpTime}, \text{ExpSpace}\}$. A problem L' is *$\mathcal{C}$-hard* if $L \leq_p L'$ for all $L \in \mathcal{C}$. It is *$\mathcal{C}$-complete* if it is in $\mathcal{C}$ and $\mathcal{C}$-hard. △

Hardness for LogSpace is defined in terms of logspace reductions. Is NP-hardness a useful notion? After all, a problem being NP-hard means that this problem can be "used" to solve *all* problems in NP.

Theorem 26. NP-*complete problems exist.*

Proof. (sketch) For M an NTM over Σ, $w \in \Sigma^*$ and $k \geq 0$, let $\langle M, w, 1^k \rangle$ be an encoding of M and w followed by a string of k times the symbol "1". Set

$$U := \{\langle M, w, 1^k \rangle \mid \text{NTM } M \text{ accepts } w \text{ in at most } k \text{ steps}\}.$$

U is in NP. We can construct a polytime-bounded NTM M_U that reads its input copying w to a separate tape and then simulates M on w by interpreting M's transition table (guessing to make non-deterministic choices). M_U accepts if M accepts. It rejects if M rejects or more than k steps were made (checked by a unary counter on another tape). This can be done in polytime.

U is NP-hard. Let $L \in$ NP. We have to show $L \leq_p U$. There is an n^k-time bounded NTM M with $L(M) = L$, for some $k \in \mathbb{N}$. The reduction function f takes input w to $\langle M, w, 1^{n^k} \rangle$. Clearly, $w \in L$ iff $\langle M, w, 1^{n^k} \rangle \in U$. Also, f can be computed in time $\mathcal{O}(n^k)$. ❑

Completeness for a complexity class seems to be a very strong condition. Indeed, the problem U is contrived since it is a "universal interpreter" for NP problems. Are there *natural* NP-complete problems? Surprisingly, it turns out that there are many.

4.5 SAT: A Natural NP-complete Problem

The first natural problem that was proved NP-complete is propositional satisfiability, i.e.,

$$\text{SAT} := \{\varphi \mid \varphi \text{ is a satisfiable formula of propositional logic}\}.$$

This is the famous theorem of Cook/Levin.

Theorem 27 (Cook/Levin). *SAT is* NP-*complete.*

Proof. In NP: given a formula φ_0 with variables $p_1, \ldots, p_n$, guess a truth assignment for $p_1, \ldots, p_n$ and check whether it satisfies φ.

NP-hardness: We have to show that $L \leq_p SAT$ for all $L \in$ NP. Fix $L \in$ NP. Then there is a (1-tape) NTM M and a $k \geq 0$ such that $L(M) = L$ and M is n^k-time bounded. We show how to construct for each input w to M in polytime a propositional formula φ_w such that φ_w is satisfiable iff M accepts w. This defines the required polytime reduction.

Our aim is to construct ψ_w such that it describes accepting computations of M on w. A computation of M on $w = a_0 \cdots a_{n-1}$ can be visualized as a matrix:

q_0, a_0	a_1	$\cdots$	a_n	$\sqcup$	$\cdots$	$\sqcup$
b	q, a_1	$\cdots$	a_n	$\sqcup$	$\cdots$	$\sqcup$
b	q', b'	$\cdots$	a_n	$\sqcup$	$\cdots$	$\sqcup$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Both the width (tape consumption) as well as the height (number of steps) is bounded by $n^k + 1$. The matrix entries can be represented using variables:

- $T_{a,i,t}$: at time t, tape cell i wears label a;
- $H_{i,t}$: at time t, the head is on cell i;
- $S_{q,t}$: at time t, M is in state q;

where $a \in \Gamma$, $q \in Q$, $t, i \leq n^k$. The following formula describes initial configurations (recall $w = a_0 \cdots a_{n-1}$):

$$\psi_{\mathsf{ini}} := S_{q_0,0} \wedge H_{0,0} \wedge \bigwedge_{i<n} T_{a_i,i,0} \wedge \bigwedge_{n \leq i \leq n^k} T_{\sqcup,i,0}.$$

Define $R(i) = i + 1$; $L(i) = i - 1$ if $i > 1$ and $L(0) = 0$. M moves according to its transition relation:

$$\psi_{\mathsf{move}} := \bigwedge_{q \in Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}, a \in \Gamma, i \leq n^k, t < n^k} \Big((S_{q,t} \wedge H_{i,t} \wedge T_{a,i,t}) \rightarrow \bigvee_{(q,a,q',a',M) \in \Delta, M(i) \leq n^k} (S_{q',t+1} \wedge H_{M(i),t+1} \wedge T_{a',i,t+1}) \Big)$$

Cells that are not underneath the head do not change:

$$\psi_{\mathsf{keep}} := \bigwedge_{a \in \Gamma, t < n^k, i \leq n^k} \big(\neg H_{i,t} \wedge T_{a,i,t} \rightarrow T_{a,i,t+1} \big)$$

M terminates successfully:

$$\psi_{\mathsf{acc}} := \bigvee_{t \leq n^k} \big(S_{q_{\mathsf{acc}},t} \wedge \bigwedge_{t' < t} \neg S_{q_{\mathsf{rej}},t'} \big)$$

There are no double labellings, double heads, etc:

$$\psi_{\mathsf{aux}} := \bigwedge_{t,q,q',q \neq q'} \neg(S_{q,t} \wedge S_{q',t}) \wedge \bigwedge_{i,t,a,a',a \neq a'} \neg(T_{a,i,t} \wedge T_{a',i,t}) \wedge \bigwedge_{t,i,j,i \neq j} \neg(H_{i,t} \wedge H_{j,t})$$

where the conjunctions range over the obvious values. Finally, set

$$\phi_w := \psi_{\mathsf{ini}} \wedge \psi_{\mathsf{move}} \wedge \psi_{\mathsf{keep}} \wedge \psi_{\mathsf{acc}} \wedge \psi_{\mathsf{aux}}.$$

It is possible to show that ψ_w is satisfiable iff M accepts w (Exercise). ❑

5 More NP-complete Problems

To prove NP-hardness of CLIQUE, we follow a different and more convenient approach.

Lemma 28. *Let $\mathcal{C}$ be a complexity class closed under polytime reductions. If L is $\mathcal{C}$-hard and $L \leq_p L'$, then L' is $\mathcal{C}$-hard.*

Thus, to show NP-hardness we can simply give a polytime reduction from any *single* NP-hard problem such as SAT. It is convenient to work with a more restricted version.

A *literal* is a variable or the negation thereof. A *clause* is a disjunction of literals. A *3-clause* is a clause containing exactly three (different) literals. A *3-formula* is a conjunction of 3-clauses. Set

$$3SAT = \{\varphi \mid \varphi \text{ is a satisfiable 3-formula}\}.$$

So an example 3-formula is $(a \vee b \vee c) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee d)$.

Theorem 29. *3SAT is* NP*-complete.*

Proof. Since SAT is in NP, 3SAT is in NP. To prove hardness, we show that SAT can be polynomially reduced to 3SAT.

Let φ be an instance of SAT, i.e., a formula. Using De Morgan's rules, we can convert φ into negation normal form in polynomial time, i.e., into an equivalent formula φ' in which negation occurs only in front of propositional letters.

Let Γ be the set of all subformulas of φ', including φ'. The cardinality of Γ is bounded by the length of φ'. Define a substitution function that replaces non-literals $\sigma \in \Gamma$ with with a fresh variable p_σ:

$$s(\sigma) := \begin{cases} \sigma & \text{if } \sigma \text{ is a literal} \\ p_\sigma & \text{otherwise} \end{cases}$$

Set

$$\psi := \bigwedge_{\sigma = \vartheta \wedge \chi \in \Gamma} (p_\sigma \leftrightarrow s(\vartheta) \wedge s(\chi)) \wedge \bigwedge_{\sigma = \vartheta \vee \chi \in \Gamma} (p_\sigma \leftrightarrow s(\vartheta) \vee s(\chi))$$

It is not hard to see that φ is satisfiable iff $\psi \wedge p_{\varphi'}$ is and that the latter can be constructed in polytime. It remains to show that any formula of the form $\ell_1 \leftrightarrow \ell_2 \wedge \ell_3$, $\ell_1 \leftrightarrow \ell_2 \wedge \ell_3$, and ℓ_1 (with ℓ_1, ℓ_2, ℓ_3 literals) can be converted in polytime into a 3-formula. We only do one case:

$$\begin{aligned} \ell_1 \leftrightarrow \ell_2 \wedge \ell_3 &\rightsquigarrow (\neg \ell_1 \vee (\ell_2 \wedge \ell_3)) \wedge (\neg(\ell_2 \wedge \ell_3) \vee \ell_1) \\ &\rightsquigarrow (\neg \ell_1 \vee \ell_2) \wedge (\neg \ell_1 \vee \ell_3) \wedge (\neg \ell_2 \vee \neg \ell_3 \vee \ell_1) \\ &\rightsquigarrow (\neg \ell_1 \vee \ell_2 \vee x) \wedge (\neg \ell_1 \vee \ell_2 \vee \neg x) \wedge \\ & (\neg \ell_1 \vee \ell_3 \vee x') \wedge (\neg \ell_1 \vee \ell_3 \vee \neg x') \wedge \\ & (\neg \ell_2 \vee \neg \ell_3 \vee \ell_1) \end{aligned}$$

where x and x' are fresh variables. ❑

Interestingly, it can be shown that 2-SAT is in P. We now reduce 3SAT to CLIQUE.

Theorem 30. *Clique is* NP*-complete.*

Proof. We have already seen that clique is in NP, so it suffices to show that 3SAT $\leq_p$ CLIQUE.

Let $\varphi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \cdots \wedge (\ell_{k,1} \vee \ell_{k,2} \vee \ell_{k,3})$ be a 3-formula. We define a graph $G_\varphi := (V, E)$ with

- $V = \{\langle i, j\rangle \mid 1 \leq i \leq k \text{ and } 1 \leq j \leq 3\}$;
- $E = \{\{\langle i, j\rangle, \langle i', j'\rangle\} \mid i \neq i' \text{ and } \ell_{i,j} \text{ does not contradict } \ell_{i',j'}\}$.

Then φ is satisfiable iff G_φ has a k-clique.

"$\Rightarrow$". Let φ be satisfiable. Then there is a truth assignment that, in each clause, makes at least one literal true. Select one true literal from every clause, generating a set of nodes $W \subseteq V$. Then W is of cardinality k and a clique in G_φ: no two elements of W are from the same clause or associated with contradictory literals.

"$\Leftarrow$". Let $W \subseteq V$ be a clique in G_φ of size k. Let Θ be the set of associated literals. Since W is a clique, Θ contains no contradictory literals and we can construct a truth assignment t that makes all literals in Θ true. For the same reason, Θ contains no two literals from the same clause. Since W is of size k, Θ contains one literal from each clause. Thus, t satisfies φ. ❑

SAT, 3SAT and CLIQUE are just three of many NP-complete problems, many of them very different in nature. We give one more example.

The subset-sum problem is to determine, given a collection of numbers $x_1, \dots, x_k \in \mathbb{N}$ and a target number $t \in \mathbb{N}$ (all in binary), whether there is a subcollection that adds up to t:

$$\text{SUBSET-SUM} := \{(S, t) \mid S = \{x_1, \dots, x_k\} \text{ and there is a } \{y_1, \dots, y_\ell\} \subseteq S \text{ such that } \textstyle\sum_{y_i} = t\}.$$

In this definition, we assume that collections are multi-sets and thus allow repetition of elements.

Theorem 31. *SUBSET-SUM is* NP*-complete.*

Proof. A polytime-bounded NTM may simply guess a sub-collection and check whether it adds up to t. This shows containment in NP.

NP-hardness is shown by reduction from 3SAT. Given a 3-formula φ with ℓ variables and k clauses, we generate an instance (S, t) of SUBSET-SUM, where S consists of two numbers y_i and z_i for each variable x_i and two numbers g_i and h_i for each clause c_i.

Intuitively, (S, t) is constructed such that every sub-collection of S summing up to t represents a satisfying assignment of φ: if the variable x_i has truth value 1, then y_i is in the sub-collection of S summing up to t, but z_i is not. If x_i has truth value 0, it is the other way around. The g_i and h_i are needed for technical purposes.

As an example, consider the 3-formula

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_2 \vee \cdots) \wedge \cdots \wedge (\neg x_3 \vee \cdots \vee \cdots).$$

We define the numbers in (S, t) as given in decimal by the following table:

	1	2	3	4	$\cdots$	ℓ	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$						$\vdots$	$\vdots$		$\vdots$	$\vdots$
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$										$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

The decimal representation of each number y_i, z_i is in two parts. Intuitively, the left part is used to ensure that exactly one of y_i, z_i is in S, for all i. The right part reflects the literals that appear in clauses: the y_i/c_j field is 1 if the literal x_i appears in clause c_j and the z_i/c_j field is 1 if the literal $\neg x_i$ appears in clause c_j.

Let φ be satisfiable and T a satisfying assignment. Define a sub-collection of S by selecting y_i if $T(x_i) = 1$ and z_i is $T(x_i) = 0$. If we add up the resulting numbers, we obtain 1s in the left-hand part as required, and numbers between 1 and 3 in the right-hand part. Select enough g_i and h_i to make all digits in the right part equal to three.

Let S' be a sub-collection of S summing up to t. Then exactly one of y_i, z_i is in S', for all i. Define a truth assignment T by setting $T(x_i) = 1$ if $y_i \in S'$ and $T(x_i) = 0$ otherwise. This assigment satisfies all clauses c_j: in column c_j, at most 2 can come from the g_i and h_i, so at least 1 must come from some y_i or z_i. Since the corresponding literal occurs in c_j, the clause is satisfied. ❏

5.1 P vs. NP

We have seen several examples for problems in NP that are hardest among all such problems. It is likely that no such problem is in P.

The problem whether P = NP is one of the most important and most difficult open problems in computer science and mathematics. We can state in in two ways:

- Using the definition: are polytime-bounded NTMs more powerful than polytime-bounded DTMs?

  But why should NTMs be so important?

- Using the alternative characterization: is *checking the existence* of a polysized proof more difficult than *verifying a given* polysized proof (both using DTMs)? Or even, *are polysized proofs useless*?

  This seems much more relevant.

P = NP seems unlikely and it is generally believed that P $\neq$ NP although no proof exists (the Clay Mathematics Institute has even set out a prize of one million dollars for a proof).

The P = NP question becomes even more important by the fact that any *single* NP-complete problem such as clique is closely tied to it. Trivially, if we can prove of any problem in NP (such as NP-complete ones) that it is not in P, then P $\neq$ NP. Additionally:

Theorem 32. *If there is an* NP*-hard problem in* P*, then* P = NP*.*

Proof. Let L be NP-hard and in P. Let $L' \in$ NP. Since L is NP-hard, $L' \leq_p L$. Since L is in P, it follows that $L' \in$ P. ❏

Thousands of problems have been shown NP-complete. Things reached a situation where people seem to expect any problem in NP to be either NP-complete or in P. This naive view is wrong.

Theorem 33 (Ladner's Theorem). *If* P $\neq$ NP*, then there is an* $L \in$ NP *such that* L *is neither in* P *nor* NP*-complete.*

A proof of this Theorem if out of scope. The basic idea is to start with an NP-complete problem such as SAT and to "punch holes into it", i.e., modifying it by removing "yes" instances in a way such that (i) all possible reductions from NP-complete problems fail but (ii) it is still so hard that any polytime algorithm fails as well.

Problems that are as in Theorem 33 are called NP*-intermediate.* Nobody has yet found a natural NP-intermediate problem (under the assumption that P $\neq$ NP). Two problems that are candidates for being NP-intermediate are:

- Graph isomorphism. Given two undirected graphs G_1 and G_2, decide whether it is possible to rename the nodes in G_2 such that $G_1 = G_2$;

- Integer factorization. Given integers i and j, does i have a factor less than j? This problem is very important for cryptography.

A final remark: the arguments we have used to show that problems are in NP were always of a trivial kind. This need not alway be the case. For example, it is possible but not trivial to show that the implication between negation-free formulas of first-order logic is in NP (and, in fact, NP-complete). Showing this is not a triviality.

5.2 The Complement of NP

For some complexity classes such as NP, it is important to carefully distinguish between the class and its complement.

Definition 34. Let $\mathcal{C}$ be a complexity class. Then co-$\mathcal{C}$ is the class of languages $L \subseteq \Sigma^*$ such that $\Sigma^* \setminus L \in \mathcal{C}$. △

In particular, co-NP is the class of all problems whose complement is in NP. This is an interesting class and we will look at it shortly. In contrast, complements of *deterministic* classes are not very interesting.

Lemma 35. *If $\mathcal{C}$ is a deterministic (time or space) complexity class, then $\mathcal{C}$ = co-$\mathcal{C}$.*

Proof. "$\subseteq$". Let $L \in \mathcal{C}$ and take a TM M witnessing this. Since M is deterministic, it has only a single computation on every input. This means that we can convert M into a maching for $\overline{L}$ (with the same time and space consumption) by simply swapping q_{acc} and q_{rej}. Thus, $\overline{L} \in \mathcal{C}$, implying $L \in$ co-$\mathcal{C}$.

"$\supseteq$". Let $L \in$ co-$\mathcal{C}$. Then $\overline{L} \in \mathcal{C}$ and reasoning as in the previous direction we get $L \in \mathcal{C}$. ❑

Observe that the proof of Lemma 35 does not work for non-deterministic TMs. If such a TM accepts an input w, it may have multiple computations on w, some of them accepting and some of them rejecting. Swapping q_{acc} and q_{rej} will not change this situation. Lemma 35 leaves us with the following (potentially) interesting complement classes:

co-NP, and co-NExpTime.

We will focus on co-NP. In terms of our alternative characterization, co-NP is the set of problems L such that there is a succinct and polytime verifiable proof of *non*-membership in L. For example,

Theorem 36. *Validity of propositional formulas is in co-*NP*.*

Proof. We have to show that non-validity is in NP. We do this by reduction to satisfiability: φ is not valid iff $\neg\varphi$ is satisfiable. ❑

For complement classes, hardness and completeness are defined as usual (in terms of polytime or logspace reductions).

Theorem 37. *L is* NP*-complete iff $\overline{L}$ is co-*NP*-complete.*

Proof. By definition, $L \in$ NP iff $\overline{L} \in$ co-NP. It thus suffices to show that L is NP-hard iff $\overline{L}$ is co-NP-hard.

Assume that $L \subseteq \Sigma^*$ is NP-hard. To prove that $\overline{L}$ is co-NP-hard, we have to show that $L' \leq_p \overline{L}$ for all $L' \in$ co-NP. Let $L' \in$ co-NP with $L' \subseteq \Gamma^*$. Then $\overline{L}' \in$ NP and thus $\overline{L}' \leq_p L$ which is witnessed by a polytime reduction $f : \Gamma^* \to \Sigma^*$ from $\overline{L'}$ to L. It is easily seen that f is also a polytime reduction from L' to $\overline{L}$, which proves $L' \leq_p \overline{L}$. The converse direction is analogous. ❑

Thus, propositional *un*satisfiability and the *co*-clique problem (given a graph G and a $k \geq 1$, does G *not* have a clique of size k?) are co-NP-complete. Validity of propositional formulas is also co-NP-complete: by Theorem 37, it suffices to show that non-validity is NP-complete. We have already shown that tis in in NP. NP-hardness follows by reduction of satisfiability: φ is satisfiable iff $\neg\varphi$ is not valid.

It is not hard to see that $\mathcal{C} \subseteq \mathcal{C}'$ implies co-$\mathcal{C}$ $\subseteq$ co-$\mathcal{C}'$. Thus co-P $\subseteq$ co-NP, i.e., P $\subseteq$ co-NP. Also, co-NP $\subseteq$ co-PSpace, i.e., co-NP $\subseteq$ PSpace.

Thus, both NP and co-NP are situated between P and PSpace. Not much is known about the exact relationship between NP and co-NP. Just like P $\neq$ NP, the following are commonly believed to be true, but proofs do not exist:

1. NP $\neq$ co-NP;
2. NP$\cap$co-NP $\neq$ P.

It is easily seen that each of these two conjectures implies P $\neq$ NP (whereas their complement does *not* imply P = NP).

6 The Polynomial Hierarchy

It is possible to identify a much richer structure between P and PSpace than just NP and co-NP. This is based on oracle Turing machines.

Definition 38. An *oracle TM* M^O is a (deterministic or non-deterministic) TM equipped with an *oracle* O, which is simply a language $O \subseteq \Sigma^*$. An oracle TM has an additional oracle tape and three distinguished states $q_?$, q_+, and q_-. For all states except $q_?$, transitions are defined as usual. If M^O enters state $q_?$, then the next state is q_+ if the word on the oracle tape belongs to O, and q_- otherwise. This transition is instantaneous, and does not change the head positions and tape contents. △

Thus, an oracle TM M^O is equipped with a "sub-procedure" for deciding O which is viewed as a black box. In particular, the time and space consumption of this "procedure" are not counted. Indeed, O does not even have to be decidable.

Definition 39.

- $\mathrm{P}^O := \{L \mid L$ is decided by an oracle DTM M^O in polynomial time$\}$;
- $\mathrm{NP}^O := \{L \mid L$ is decided by an oracle NTM M^O in polynomial time$\}$;
- For a complexity class $\mathcal{C}$, $\mathrm{P}^{\mathcal{C}} := \bigcup_{L \in \mathcal{C}} \mathrm{P}^L$ and $\mathrm{NP}^{\mathcal{C}} := \bigcup_{L \in \mathcal{C}} \mathrm{NP}^L$.

△

For example, a problem L is in P^{NP} if there exists a polytime-bounded oracle DTM M^O whose oracle O is in NP and such that $L(M^O) = L$. Note that we cannot show that NP = P^{NP} by simply joining M^O and a polytime-bounded NTM M for O into a new NTM N: to be sure that it should switch to q_-, a single computation of N would

have to analyse *all* computations of M, which does not seem possible. In contrast, $\mathrm{P}^{\mathrm{P}} = \mathrm{P}$ is easily shown along these lines.

We can now define more complexity classes between P and PSpace: the polynomial hierarchy.

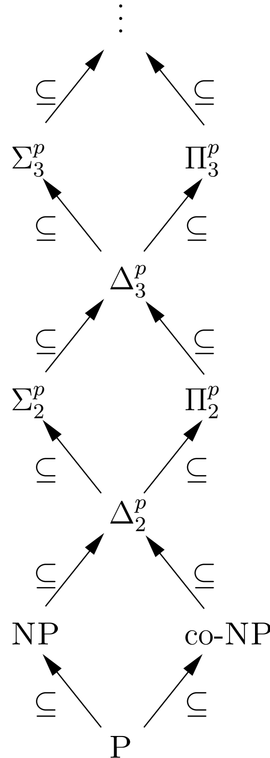
Definition 40. For $k \geq 0$, we inductively define the complexity classes Σ_k^p, Π_k^p, and Δ_k^p as follows:

- $\Sigma_0^p := \Pi_0^p := \Delta_0^p := \mathrm{P}$
- $\Sigma_{k+1}^p := \mathrm{NP}^{\Sigma_k^p}$
- $\Pi_{k+1}^p := \text{co-}\Sigma_{k+1}^p$
- $\Delta_{k+1}^p := \mathrm{P}^{\Sigma_k^p}$

△

It is not very difficult to prove that $\Delta_1^p = \mathrm{P}$ (since $\mathrm{P}^{\mathrm{P}} = \mathrm{P}$), $\Sigma_1^p = \mathrm{NP}$, and $\Pi_1^p = \text{co-NP}$.

It is possible to derive the following inclusions:



There is also a complexity class for the whole polynomial hierarchy:

Definition 41. $PH := \bigcup_{k \geq 0} \Sigma_k^p$. △

Obviously, $\mathrm{P} \subseteq PH$. It is also possible to show that $PH \subseteq$ PSPACE. It is not known whether any of the inclusions shown above is strict. In particular, it is not known whether the polynomial hierarchy *collapses*, i.e., whether there is a $k \geq 0$ such that $PH = \Sigma_k^p$. However, the following are known:

- If P = NP, then $\mathrm{P} = PH$, i.e., the polynomial hierarchy collapses to P;
- If $PH =$ PSPACE, then the polynomial hierarchy collapses.

Hardness for all these complexity classes is defined in terms of polytime reductions. We give an example for a Π_2^p-complete problem.

In propositional logic, we may view a truth assignment t as the set of all those variables that are true under the assignment. A *model* of a propositional formula φ is a truth assignment t that satisfies φ. A model of φ is called *minimal* if all truth assignments $t' \subsetneq t$ do not satisfy φ. The *minimal implication problem* is to decide, given two formulas φ and ψ, whether all minimal models for φ are models of ψ.

We only show that minimal implication is in $\Pi_2^p = \text{co-NP}^{\text{NP}}$. As a preliminary, we show that the following problem is in NP: given a formula φ and a truth assignment t, check whether there is a model t' of φ with $t' \subsetneq t$. Let us call this problem L. L is in NP since a polytime NTM may simply guess an assignment t' and then check whether it is as required.

Now we show that the complement of minimal implication is in NP^{NP}: given φ and ψ, a polytime NTM may guess a truth assignment t and verify that (1) t is a model of φ and of $\neg\psi$ and (2) t is *minimal*. To do the latter, it calls an oracle for L with input t and φ and negates the result.

Note (again) that the two NTMs cannot be easily joined into one: since we negate the result returned by the oracle, we would have to consider *all* its computations.

Minimal implication is also hard for Π_2^p, but we do not have time to prove that. A large source for problems that are complete for classes in the polynomial hierarchy comes from so-called non-monotonic logics.

7 PSpace

Recall that PSpace-completeness is defined similarly to NP-completeness, i.e., in terms of polytime reductions. The prototypical PSpace-complete problem is the validity of quantified Boolean formulas (QBFs).

Definition 42. A *quantified Boolean formula (QBF)* is of the form

$$Q_1 p_1 \cdots Q_n p_n . \varphi$$

where $Q_i \in \{\forall, \exists\}$ and φ is a formula of propositional logic using only variables $p_1, \ldots, p_n$. △

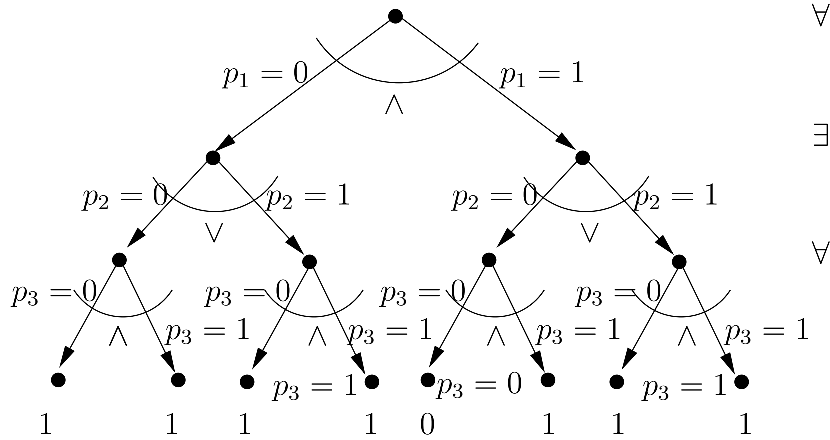
Validity is defined via induction on the length of the quantifier prefix. If φ is a propositional formula, then $\varphi[p/1]$ denotes φ with p replaced by 1 (logical truth), and similarly for $\varphi[p/0]$.

Definition 43 (QBF Validity). A QBF formula $Q_1 p_1 \cdots Q_n p_n \varphi$ is *valid* iff

1. $Q_1 = \exists$ and one of $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/0]$ and $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/1]$ is valid;
2. $Q_1 = \forall$ and both $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/0]$ and $Q_2 p_2 \cdots Q_n p_n \varphi[p_1/1]$ are valid.
3. $n = 0$ and φ (which then does not contain any variables) evaluates to true.

△

Consider the QBF $\psi = \forall p_1 \exists p_2 \forall p_3 \,.\, p_1 \to p_2 \vee p_3$. The validity of ψ can be checked using an and-or-tree:



Observe that each level of the tree corresponds to one of the quantifiers, ∀-levels use ∧-branching and ∃-levels use ∨-branching.

Define

$$QBF := \{\varphi \mid \varphi \text{ is a valid quantified Boolean formula}\}.$$

Although quantification trees are of exponential size, QBF can be decided in PSpace.

Theorem 44. *QBF is in* PSpace.

Proof. The following recursive procedure decides validity of QBFs:

$\mathsf{Valid}(Q_1p_1 \cdots Q_np_n.\varphi)$
  if $n = 0$ then
    $\psi = \varphi$ is a propositional formula over $\{0,1\}$
    evaluate ψ and return the result
  endif
  set $v := \mathsf{Valid}(Q_2p_2 \cdots Q_np_n.\varphi[p/0])$ and $v' := \mathsf{Valid}(Q_2p_2 \cdots Q_np_n.\varphi[p/1])$
  if ψ begins with $\exists p$ then
    return $v \vee v'$
  else (ψ begins with $\forall p$)
    return $v \wedge v'$
  endif

The recursion depth is bounded by the length of the quantifier prefix. For each recursive call, one needs to store the current QBF and whether the purpose of the call was to compute v or v'. In total, this requires quadratic space. ❑

It is easy to show that QBF is NP-hard by reduction from SAT: a propositional formula φ in variables $p_1, \ldots, p_n$ is satisfiable iff $\exists p_1 \cdots \exists p_n.\varphi$ is valid. Indeed, QBF is even PSpace-hard.

Theorem 45. *QBF is* PSpace*-hard, thus* PSpace*-complete.*

Proof. We show that $L \leq QBF$ for all $L \in$ PSpace. Fix $L \in$ PSpace. Then there is a $d \geq 0$ and an n^d-space bounded (1-tape) DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ such that $L(M) = L$. We construct in polytime for each input w a QBF ψ_w such that M accepts w iff ψ_w is valid. Since M may need exponential time, we cannot use the same proof as in Cook's theorem, e.g. introducing variables $S_{q,t}$ saying that M is in state q at time t.

Instead, we basically reformulate the proof of Savitch's theorem "in the language of logic". That proof was based on a predicate $\mathsf{Path}(c, c', i)$ expressing that there is a path of length at most 2^i from c to c' in $G_M(n^d)$. A configuration is represented by a tuple of the following variables:

- S_q for every $q \in Q$ describes the state;
- $T_{a,i}$ for every $a \in \Gamma$ and $i \leq n^d$ describes the symbol on the i-th tape cell;
- H_i for every $i \leq n^d$ describes the head position.

The following formula ensures that such a tuple $\overline{C}$ encodes a valid configuration:

$$\begin{aligned}\psi_{\mathsf{conf}}(\overline{C}) \;:=\; & \bigvee_{q \in Q} S_q \wedge \bigwedge_{q,q' \in Q, q \neq q'} \neg(S_q \wedge S_{q'}) \wedge \bigvee_{i \leq n^d} H_i \wedge \bigwedge_{i,i' \leq n^d, i \neq i'} \neg(H_i \wedge H_{i'}) \wedge \\ & \bigwedge_{i \leq n^d} \bigvee_{a \in \Gamma} T_{a,i} \wedge \bigwedge_{i \leq n^d, a, a' \in \Gamma, a \neq a'} \neg(T_{a,i} \wedge T_{a',i})\end{aligned}$$

The description of $\mathsf{Path}(c, c', i)$ is based on steps of M as described by the following formula (where $L(i)$ and $R(i)$ is defined as in the proof of Cook's theorem):

$$\begin{array}{rcl}\psi_{\mathsf{next}}(\overline{C}, \overline{C}') & := & \psi_{\mathsf{conf}}(\overline{C}) \wedge \psi_{\mathsf{conf}}(\overline{C}') \wedge \\ & & \bigwedge\limits_{i \leq n^d} \Big(H_i \rightarrow \Big(\bigwedge\limits_{j \leq n^d, j \neq i, a \in \Gamma} (T_{a,j} \leftrightarrow T'_{a,j}) \wedge \\ & & \qquad \bigwedge\limits_{\delta(q,a)=(q',a',M), M(i) \leq n^d} (S_q \wedge T_{a,i}) \rightarrow (S'_q \wedge T'_{b,i} \wedge H'_{M(i)}) \Big) \Big)\end{array}$$

We can define $\mathsf{Path}(c, c', i)$ for the case $i = 0$ as follows:

$$\psi^0_{\mathsf{reach}}(\overline{C}, \overline{C}') := \psi_{\mathsf{eq}}(\overline{C}, \overline{C}') \vee \psi_{\mathsf{next}}(\overline{C}, \overline{C}')$$

where $\psi_{\mathsf{eq}}(\overline{C}, \overline{C}')$ is a formula saying that $\overline{C}$ and $\overline{C}'$ are identical. To define the case $i > 0$, it is tempting to put

$$\psi^i_{\mathsf{reach}}(\overline{C}, \overline{C}') := \exists \overline{C}''.\psi^{i-1}_{\mathsf{reach}}(\overline{C}, \overline{C}'') \wedge \psi^{i-1}_{\mathsf{reach}}(\overline{C}'', \overline{C}')$$

but this involves nested duplication and implies that the length of ψ^i_{reach} is at least 2^i. The way out is provided by universal quantifiers:

$$\begin{array}{rcl}\psi^i_{\mathsf{reach}}(\overline{C}, \overline{C}') & := & \exists \overline{C}'' \forall \overline{K} \forall \overline{K}'.\big(\ (\psi_{\mathsf{eq}}(\overline{C}, \overline{K}) \wedge \psi_{\mathsf{eq}}(\overline{C}'', \overline{K}')) \vee (\psi_{\mathsf{eq}}(\overline{C}'', \overline{K}) \wedge \psi_{\mathsf{eq}}(\overline{C}', \overline{K}')) \\ & & \qquad \rightarrow \psi^{i-1}_{\mathsf{reach}}(\overline{K}, \overline{K}') \ \big)\end{array}$$

It remains to devise formulas $\psi^w_{\mathsf{input}}(\overline{C})$ expressing that $\overline{C}$ is the initial configuration on input w and $\psi_{\mathsf{acc}}(\overline{C})$ saying that $\overline{C}$ is accepting (left as exercise).

If M is n^d-space bounded, then it is $T(n) = 2^{c \cdot n^d}$-time bounded for some $c \geq 1$. The final reduction formula is:

$$\psi_w := \exists \overline{C} \exists \overline{C}'.(\ \psi^w_{\mathsf{input}}(\overline{C}) \wedge \psi_{\mathsf{acc}}(\overline{C}) \wedge \psi^{T(|w|)}_{\mathsf{reach}}(\overline{C}, \overline{C}') \ )$$

Strictly, ψ_w is not a QBF since the quantifiers do not occur as a prefix. However, it is possible to "pull out" quantifiers (exercise). ❑

Typical PSpace-hard problems include generalizations of games such as "given an $n \times n$ checkerboard with some red kings and some blackkings on it and the player whose turn it is, does black have a winning strategy?". Also in formal languages, automata theory, and logic there are many PSpace-complete problems. An example for the first group is universality of regular expressions, i.e., "given a regular expression π, is $L(\pi) = \Sigma^*$?

It is possible to modify the QBF problem such that a complete problem for the complexity classes Σ^p_k and Π^p_k of the polynomial hierarchy is obtained. In the following, we abbreviate $\exists x_1.\exists x_2.\cdots \exists x_n.\varphi$ with $\exists \overline{X}.\varphi$, and likewise for universal quantifiers. Then, a Σ_k-*QBF* is a QBF of the form

$$\exists \overline{X}_1.\forall \overline{X_2}.\cdots Q_k \overline{X}_k.\varphi$$

i.e., the quantifier prefix consists of k alternating blocks of existential and universal quantifiers, starting with an existential block. We define Π_k-*QBFs* analogously, but starting with a universal block.

Theorem 46. *For all $k \geq 1$, Σ_k-QBF is Σ_k^p-complete and Π_k-QBF is Π_k^p-complete.*

Exercise Sheet 1

for the lecture

Complexity Theory

Exercise 1

Words over the alphabet $\Sigma = \{0, 1\}$ can be viewed as the binary representation of a non-negative integer (lowest bit rightmost, possibly with additional leading zeros). Construct a 1-tape DTM that works over the alphabet Σ and does the following: when started on an input word w representing the number $n \in \mathbb{N}$, it converts w into the binary representation of $n + 1$ and stops in an accepting state. To describe the transition function, draw the state diagram rather than using tuple notation.

Exercise 2

A *palindrome* is a word that reads the same backwards, e.g. "reittier" or "malayalam". Find a k-tape DTM (for some self-chosen k) that works over the alphabet $\Sigma = \{a, b\}$ and decides whether its input is a palindrome. The first tape is the input tape (and thus cannot be modified). On every other tape, at most $\mathcal{O}(\log(n))$ cells may be used. It suffices to describe the machine on an intuitive level.

Exercise 3

Let a 1*-DTM be a DTM that is equipped with a single *two-side* infinite tape. Show that every language decided by a 1*-DTM is also decided by a 1-DTM within the same time and space bounds (up to a constant).

Exercise 4

For f, g function from $\mathbb{N}$ to $\mathbb{N}$, we write $f \in \Theta(g)$ if $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$. Intuitively, $f \in \Theta(g)$ if f and g have the same rate of growth. Show the following: The rate of growth of a polynomial is captured by the polynomial's first non-zero term, i.e.: if $p(n)$ is a polynomial of degree d, then $p(n) \in \Theta(n^d)$.

Exercise 5

Complete Theorem 3 by showing that $(d^{T(n)+1} - 1) \cdot c \cdot T(n) \in 2^{\mathcal{O}(T(n))}$.

Exercise Sheet 2

for the lecture

Complexity Theory

Exercise 1

Prove Lemma 24: All $\mathcal{C} \in \{\text{P}, \text{NP}, \text{PSpace}, \text{ExpTime}, \text{NExpTime}, \text{ExpSpace}\}$ are closed under polytime reductions, i.e., if $L' \in \mathcal{C}$ and $L \leq_p L'$, then $L \in \mathcal{C}$

Exercise 2

Finish the proof of Theorem 27 by showing that ψ_w is satisfiable iff M accepts w.

Exercise 3

Prove Lemma 28: If $\mathcal{C}$ is a complexity class closed under polytime reductions, then L being $\mathcal{C}$-hard and $L \leq_p L'$ implies that L' is $\mathcal{C}$-hard.

Exercise 4

HALF-CLIQUE is the problem to decide, given a graph G with $n \geq 1$ nodes, whether G contains a clique of size $n/2$. Prove that HALF-CLIQUE is NP-complete (Hint: to show NP-hardness, use a reduction from CLIQUE).

Exercise 5

SET-SPLITTING is the following problem: given a set S and a collection $C = S_1, \ldots, S_k$ of subsets of S, for some $k > 0$, decide whether all elements of C can be colored *red* or *black* such that no S_i has all its elements colored with the same color. Show that SET-SPLITTING is NP-complete (Hint: to show NP-hardness, use a reduction from 3SAT).

Foundations Exam

for the lecture

Complexity Theory

in Winter Semester 2006/2007

Exercise 1
For each of the following statements, say whether it is true or false. Justify your answers.

1. 3SAT is in PSpace;
2. SUBSET-SUM $\leq_p$ CLIQUE;
3. LogSpace $\neq$ ExpTime;
4. if $L \leq_p L'$ and L' is NP-hard, then L is NP-hard;

Exercise 2
A *Hamilton path* in an undirected graph $G = (V, E)$ is a path that visits each node exactly once. More precisely, a Hamilton path in G is a sequence of distinct vertices $v_0, \dots, v_{n-1}$ such that $\{v_0, \dots, v_{n-1}\} = V$ and $\{v_i, v_{i+1}\} \in E$ for all $i < n$.

Argue that the following problem is in NP: given a graph $G = (V, E)$, decide whether there is a Hamilton path in G.

Exercise 3
A *system of inequalities* S is a finite set of inequalities of the form

$$c_1 \cdot v_1 + c_2 \cdot v_2 + \cdots + c_n \cdot v_n \sim d$$

where the c_i are non-negative integer coefficients, d is a non-negative integer, the v_i are variables, and $\sim$ is one of $=$, $\neq$, $<$, $\leq$, $\geq$, and $>$. A *solution* to S is an assignment of non-negative integer values to variables such that all inequalities are satisfied.

Use a reduction from 3SAT to prove that the following problem is NP-hard: given a system of inequalities, decide whether it has a solution.