Perry Hart
CIS 511
Spring 2019

Abstract

These notes are based on Anindya De's "Theory of Computation" lectures given at UPenn along with Michael Sipser's *Introduction to the Theory of Computation, 3rd ed.* and Arora and Barak's *Computational Complexity: A Modern Approach*. Any mistake in what follows is my own.

Contents

1 Automata theory

1.1 Lecture 1

Definition 1.1.1.

1. An *alphabet* is a nonempty finite set of characters, e.g., $\Sigma \coloneqq \{0, 1\}$.
2. A *string* is a finite ordered sequence of elements from a given alphabet Σ. The empty sequence ϵ is allowed.
3. Let Σ^* denote the set of all finite-length strings over Σ. Any subset of Σ^* is called a *(formal) language*.

Example 1.1.2. Both the set of binary strings representing prime numbers and the set of binary strings with an even number of 1's are languages.

Remark 1.1.3. Consider a function $f : \{0, 1\}^* \to \{0, 1\}$. The computation of f is equivalent to determining whether $x \in \underbrace{f^{-1}}_{\text{a language}} \subset \{0, 1\}^*$. Thus, computing any boolean function is the same as determining membership in some language.

Note 1.1.4. Finite automata are characterized by $O(1)$ memory and passing over their inputs exactly once.

Definition 1.1.5. Formally, an *m-state deterministic finite automaton* (DFA) is an ordered 5-tuple

$$M \coloneqq (Q, \Sigma, q_0, \delta, Q_F)$$

where $|Q| = m$, Σ is an alphabet, $q_0 \in Q$, $\delta : Q \times \Sigma \to Q$, and $Q_F \subset Q$. We call δ the *transition function of M*.

Intuitively, an automaton M is a DFA if

(a) M has a finite number of states Q,

(b) M has a unique *starting state* q_0,

(c) for every state q and every symbol $\sigma \in \Sigma$, there is a unique next state $\delta(q, \sigma)$,

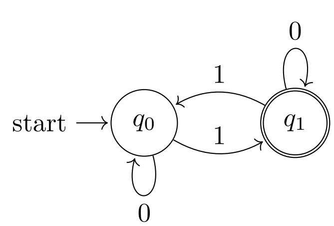
(d) computation begins at the starting state and applies δ in order, and

(e) certain states Q_F are designated as *final states*.

Definition 1.1.6. Let $x \coloneqq x_1 x_2 \cdots x_n \in \Sigma^*$. Set $q_0(x) = q_0$. For each $1 \leq i \leq n$, define $q_i(x) = \delta(q_{i-1}(x), x_i)$. If $x = \epsilon$, then $n = 0$, so that $q_0(x) = q_0$ as well. We say that x is *accepted by M* if $q_n(x) \in Q_F$. We define $L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$.

Example 1.1.7. Set $\Sigma = \{0, 1\}$.

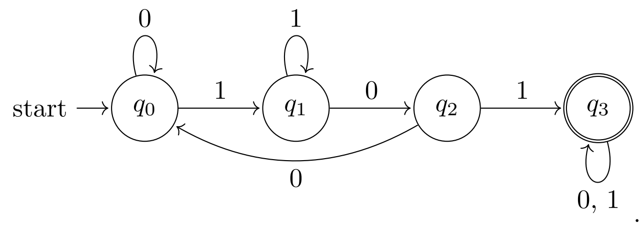
1. Let M denote



.

Then $L(M)$ consists of all binary strings with an even number of 0's.

2. Let M denote



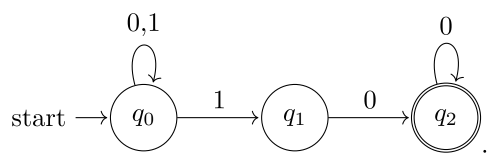
Then $L(M) = \{x \in \Sigma^* : x = y101z \text{ for some strings } y \text{ and } z\}$.

Definition 1.1.8. A language L is *regular* is there is some DFA M such that $L(M) = L$.

Remark 1.1.9. Every regular expression induces a DFA, and vice versa. Thus, they have equal expressive power. The former gives rules for generating legitimate strings whereas the latter recognizes membership of a language.

Definition 1.1.10. A *nondeterministic finite automaton* (NFA) is an ordered quintuple $(Q, \Sigma, q_0, \delta, Q_F)$ of the sort above except that $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

Example 1.1.11. Set $\Sigma = \{0, 1\}$ and $M =$



If $x = 0100$, then

$$
\begin{aligned}
q_0(x) &= \{q_0\} \\
q_1(x) &= \{q_0\} \\
q_2(x) &= \{q_0, q_1\} \\
q_3(x) &= \{q_0, q_2\} \\
q_4(x) &= \{q_0, q_2\}.
\end{aligned}
$$

1.2 Lecture 2

Definition 1.2.1. Let δ denote a transition function for the NFA N. Define the *multi-step transition function*

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$

inductively as follows.

$$
\begin{aligned}
\hat{\delta}(q, \epsilon) &= \{q\} \\
\hat{\delta}(q, x) &= \bigcup_{\gamma \in \hat{\delta}(q,y)} \delta(\gamma, \sigma) \qquad x = y\sigma,\ y \in \Sigma^*,\ \sigma \in \Sigma.
\end{aligned}
$$

Note 1.2.2. If $p \in \hat{\delta}(q, y)$ and $r \in \hat{\delta}(p, \sigma)$, then $r \in \hat{\delta}(q, y\sigma) = \hat{\delta}(q, x)$.

Definition 1.2.3. A string x is *accepted by* N if $\hat{\delta}(q_0, x) \cap Q_F \neq \emptyset$. Let $L(N) \coloneqq \{x \in \Sigma^* : N \text{ accepts } x\}$.

Remark 1.2.4. Every DFA is an NFA, in which case we have that $\hat{\delta}(q, x) = \{\delta(\hat{\delta}(q, y), \sigma)\}$. It's not the case, however, that any NFA is a DFA.

Theorem 1.2.5. *For any* NFA N*, there is some* DFA M *such that* $L(N) = L(M)$.

Proof. Write $N = (Q, \Sigma, q_0, \delta_N, Q_F)$. Define the DFA

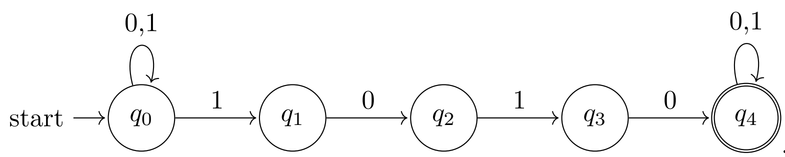
$$M = \left(\mathcal{P}(Q), \Sigma, q_0^{(1)}, \delta_M, Q_F^{(1)}\right)$$

where $q_0^{(1)} = \{q_0\}$, $\delta_M(Q', \sigma) = \bigcup_{\gamma \in Q'} \delta_N(\gamma, \sigma)$ and $Q_F^{(1)} := \{R \subset Q : R \cap Q_F \neq \emptyset\}$. For any string x, one can use induction on $|x|$ to show that if $R \subset Q$, then

$$\tilde{\delta}_M(R, x) = \bigcup_{p \in R} \widehat{\delta_N}(p, x)$$

where $\tilde{\delta}_M : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ denotes the obvious extension of δ_M to strings. By setting $R = \{q_0\}$, we are done. $\square$

Example 1.2.6. Let $L \subset \{0,1\}^*$ consist of those strings x such that "1010" appears in x. We can easily capture L with the following NFA, but writing a DFA that captures L is much harder.



.

Definition 1.2.7. An NFA is called an ϵ-NFA if its transition function is of the form $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$. In this case, we call δ an *ϵ-transition.*

Definition 1.2.8. Let q be a state. The *ϵ-closure of* q is the set of states that can be reached from q by taking finitely many ϵ-transitions. This determines a function $\epsilon\text{-cl}(-) : \mathcal{P}(Q) \to \mathcal{P}(Q)$.

Note 1.2.9. We have that $\hat{\delta}(q, x) = \begin{cases} \epsilon\text{- cl}(q) & x = \epsilon \\ \bigcup_{r \in \hat{\delta}(q,y)} \epsilon\text{- cl}(\delta(r, \sigma)) & x = y\sigma \end{cases}$.

Theorem 1.2.10. *For any ϵ-NFA N, there is some DFA M such that $L(N) = L(M)$.*

Proof. Use a similar argument to the proof for an NFA. In particular, set

$$q_0^{(1)} = \epsilon\text{- cl}(q_0)$$

and

$$\begin{aligned} \delta_M(R, \sigma) &= \bigcup_{r \in R} \bigcup_{p \in \epsilon\text{- cl}(r)} \bigcup_{s \in \delta_N(p, \sigma)} \epsilon\text{- cl}(s) \\ &= \bigcup_{r \in R} \bigcup_{s \in \delta_N(r, \sigma)} \epsilon\text{- cl}(s). \end{aligned}$$

$\square$

1.3 Lecture 3

Proposition 1.3.1. *Let $L_1, L_2 \subset \Sigma^*$ be regular.*

(a) $\overline{L_1} := \Sigma^* \setminus L_1 = (L_1)^c$ *is regular.*

Corollary 1.3.2. *If L is finite or cofinite, then it is regular.*

(b) $L_1 \cup L_2$ *is regular.*

(c) $L_1 \cap L_2$ *is regular.*

(d) *Define* $L_1 \cdot L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$*. Then* $L_1 \cdot L_2$ *is regular.*

Proof. By assumption, there exist DFA's M_1 and M_2 such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$.

(a) Construct a new DFA M_3 by making every final state of M_1 non-final and vice versa. Then $L(M_3) = \overline{L_1}$.

(b) Construct an ϵ-NFA M_3 as follows. Take a starting state q_0. Attach an ϵ-transition from q_0 to the starting state of M_1 and an ϵ-transition from q_0 to the starting state of M_2. Then $L(M_3) = L_1 \cup L_2$.

(c) Note that $L_1 \cap L_2 = (\overline{L_1} \cup \overline{L_2})^c$. Now apply (a) and (b).

(d) Construct an ϵ-NFA M_3 as follows. Given any final state q of M_1, add an ϵ-transition from q to the start state of M_2. Then $L(M_3) = L_1 \cdot L_2$.

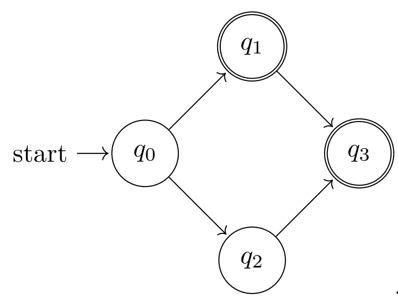
$\square$

Definition 1.3.3. Let L be any language. Let $L^k := \underbrace{L \cdot L \cdots L}_{k \text{ times}}$ for each integer $k \geq 1$. Moreover, define $L^0 = \{\epsilon\}$ (which is regular). Then define

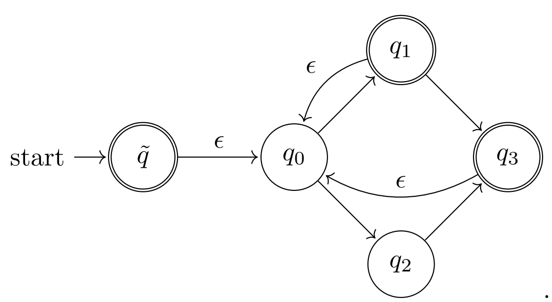
$$L^* = \bigcup_{k \geq 0} L^k.$$

Proposition 1.3.4. *If L is regular, then L^* is regular as well.*

Proof. There is some DFA M such that $L(M) = L$. Without loss of generality, write M as



Now, let $\widetilde{M}$ denote the automaton



Then $L(\widetilde{M}) = L^*$. $\square$

Remark 1.3.5. There exists a canonical set isomorphism $\{F : F \text{ is a finite automaton.}\} \cong \mathbb{N}$. Also, we have that $\{0,1\}^* \cong \mathbb{N}$. But then $\mathbb{R} \cong \mathcal{P}(\mathbb{N}) \cong \mathcal{P}(\{0,1\}^*) = \{L : L \text{ is a language over } \{0,1\}\}$. Since there is a surjection

$$\{F : F \text{ is a finite automaton.}\} \to \{L : L \text{ is a regular language over } \{0,1\}\},$$

it follows that there are uncountably many non-regular languages over $\{0,1\}$.

1.4 Lecture 4

Lemma 1.4.1 (Pumping). *Let L be a regular language. Then there exists $n_0 \in \mathbb{N}$ such that for any $x \in L$ with $|x| \geq n_0$, we may write $x = wyz$ such that*

(a) $|y| > 0$,

(b) $|wy| \leq n_0$, *and*

(c) if $i \geq 0$, *then* $wy^i z \in L$ *where* $y^i := \underbrace{y \cdots y}_{i \text{ times}}$.

In this case, we call the minimal such n_0 the pumping length of L.

Proof. By assumption, there is some DFA $M = (Q, \Sigma, \delta, q_0, Q_F)$ such that $L = L(M)$. Let $n_0 = |Q|$. Let $x \in L$ such that $|x| \geq n_0$. Set $q_i = \hat{\delta}(q_0, x_0 \cdots x_i)$ for each $i \geq 0$. There exist $0 \leq i < j \leq n_0$ such that $q_i = q_j$. Define the three strings

$$w = x_1 \cdots x_i \quad y = x_{i+1} \cdots x_j \quad z = x_{j+1} \cdots x_m$$

where $|x| = m$. It is straightforward to verify that these satisfy conditions (a), (b), and (c). $\square$

Corollary 1.4.2.

1. *Our last proof shows that any regular language L has pumping length $\leq |Q|$.*

2. *If $L(M) \neq \emptyset$, then there exists $x \in L(M)$ such that $|x| \leq |Q|$.*

Example 1.4.3. Let $n_0 \geq 0$ be an integer. Suppose that $x = wyz$ with $|y| > 0$ and $|wy| \leq n_0$.

1. Define $L = \{1^{2^n} : n \geq 0\}$. Let $x := 1^{2^{n_0+1}}$. But note that $|wy^i z| = |wyz| + (i-1)|y| = 2^{n_0+1} + (i-1)|y|$. Hence if $i = 2$, then $|wy^i z| = 2^{n_0+1} + |y| \leq 2^{n_0+1} + n_0 < 2^{n_0+2}$ since $n_0 < 2^{n_0+1}$, in which case $2^{n_0+1} < |wy^i z|$ as well. Therefore, L is not a regular language.

2. Define $L = \{ww : w \in \{0,1\}^*\}$. Let $x := 0^{n_0}10^{n_0}1$. If $|y| = m$ with $0 < m \leq n_0$, then $wz = 0^{n_0-m}10^{n_0}1$, which does not belong to L. Hence L is not a regular language.

Aside. Let D be a DFA with $|Q_D| = n$. Then D recognizes an infinite language if and only if it accepts some string s such that $n \leq |s| \leq 2n$.

Proof. The ($\Longleftarrow$) direction follows from the pumping lemma. Conversely, suppose that $L(D)$ is infinite. Then D contains some path p of states from the start state to a final state as well as some cycle of states c such that $c \cap p \neq \emptyset$. Note that $|c| \leq n$ and $|p| \leq n$. Hence we can apply c sufficiently many times to get our desired string. $\square$

2 Computability theory

Definition 2.0.1. A *Turing machine* is a 7-tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, Q_F, Q_R)$$

where $\Gamma \supset \Sigma$ is finite such that there is some *null character* $\perp \in \Gamma \setminus \Sigma$, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, and $Q_F, Q_R \subset Q$ such that $Q_F \cap Q_R = \emptyset$. We call Σ the *input alphabet*, Γ the *tape alphabet*, Q_F the set of *accepting states*, and Q_R the set of *rejecting states.*

Note 2.0.2. On any given input x, a TM can either *accept* or *reject* or *loop* (i.e., fail to halt on x).

Remark 2.0.3. A Turing machine is supposed to act as a minimal model of computation. It should be able to write, be able to move left and right, and have unconstrained memory.

2.1 Lecture 5

Remark 2.1.1. Every finite automaton may be viewed as a Turing machine M with the following properties.

(a) M never writes on the tape.

(b) M's read-write head moves to the right only.

(c) M writes on just a finite portion of the tape.

(d) M either accepts or rejects immediately after reading the input string.

Remark 2.1.2. Adding a stay option S to the set $\{L, R\}$ would not increase a Turing machine's computational power.

Definition 2.1.3. Let M be a Turing machine. Let $q \in Q$ and $u, v \in \Gamma^*$. We say that the *configuration of* M is uqv if

(a) the current state of M is q,

(b) the current tape contents is precisely uv, and

(c) the current head location is the first symbol of v.

We call this an *accepting configuration* if $q \in Q_F$ and a *rejecting configuration* if $q \in Q_R$.

Definition 2.1.4. Let $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$. Let $p, q \in Q$. We say that the configuration C_1 *yields* the configuration C_2 in the following cases.

(a) $uapbv$ yields $uqacv$ when $\delta(p, b) = (q, c, L)$.

(b) $uapbv$ yields $uacqv$ when $\delta(p, b) = (q, c, R)$.

We write $C_1 \vdash C_2$.

Definition 2.1.5. We say that the TM M *accepts* the input w if there is some sequence $C_1, \ldots, C_k$ of configurations such that C_1 is $\perp q_0 w$, each C_i yields C_{i+1} (in which case we write $C_1 \overset{*}{\vdash} C_k$), and C_k is an accepting configuration. We define the *language of the* TM M as

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

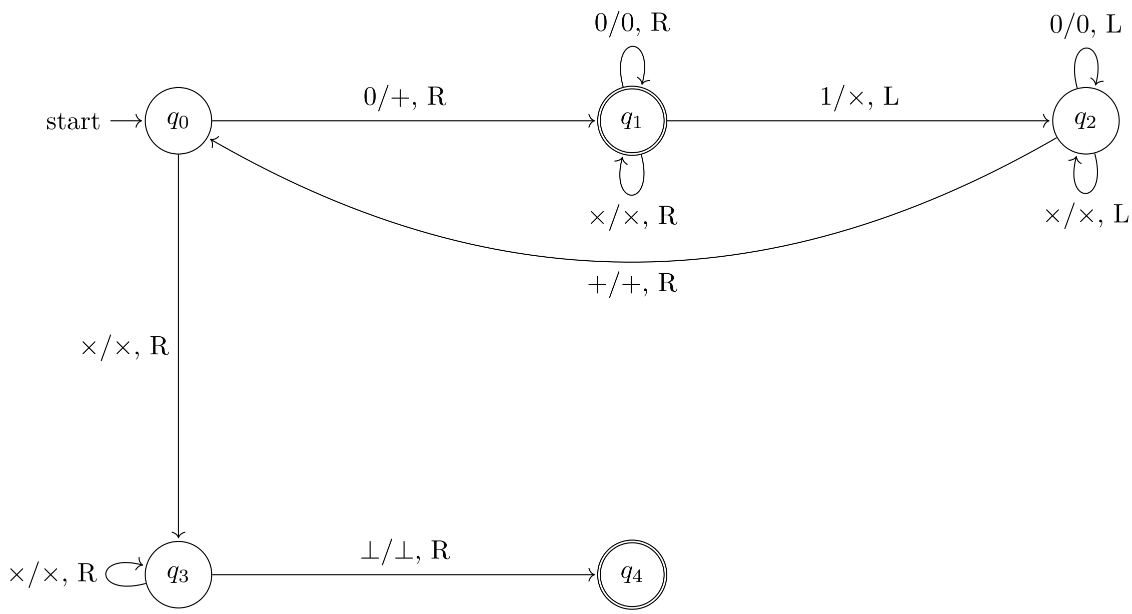
Definition 2.1.6. Let L be a language.

1. We say that L is *Turing-recognizable* or *recursively enumerable* if there is some TM M such that $L = L(M)$.
2. We say that L is *decidable* or *recursive* if there is some TM M such that $L = L(M)$ and M halts on every input. In this case, we say that M is an *algorithm*.

Note 2.1.7. Every decidable language is recursively enumerable.

Example 2.1.8. By the pumping lemma, one may show that the language $L \coloneqq \{0^n 1^n : n \geq 1\}$ is not regular. But L is decidable.

Proof. Set $\Sigma = \{0, 1\}$ and $\Gamma = \Sigma \cup \{\perp, +, \times\}$. Define the TM M as follows.



.

Then $L(M) = L$. □

Remark 2.1.9. The *Church-Turing thesis* states that our pre-theoretic notion of algorithm is entirely captured by decidability (equivalently, λ-computability).

2.2 Lecture 6

Definition 2.2.1. A *multi-tape Turing machine* is exactly like an ordinary Turing machine except that the former's transition function is of the form

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

where $k \in \mathbb{N}$.

Theorem 2.2.2. *For any $k \in \mathbb{N}$ and any language L, if there is some k-tape* TM *M such that $L(M) = M$, then there is some single-tape* TM *M' such that $L = L(M')$. Moreover, T steps of a k-tape* TM *can be simulated using $O_k(T^2)$ steps of a single-tape* TM.

Proof. Write $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_F, Q_R)$. Let

$$\{\#\} \bigcup_{\gamma \in \Gamma} \{\gamma, \dot{\gamma}\}$$

be the tape alphabet of M'. Construct M' so that its tape always has # in a cell separating the current contents of M's different tapes and $\dot{\gamma}$ whenever the head of the tape of M containing γ is currently at γ. We make M' scan its tape once to determine the positions of the heads of M's tapes, then scan it again to update its contents according to δ. □

Example 2.2.3. Let $L = \{w \in \{0, 1\}^* \mid w \text{ is a palindrome}\}$. This cannot be recognized by a TM running in better than quadratic time but can be recognized by a 2-tape TM running in linear time. Thus, computational models may differ in complexity even when they don't in decidability.

Definition 2.2.4. A *nondeterministic Turing machine* M is exactly like an ordinary Turing machine except that M's transition function is of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

such that M accepts on an input as long as at least one branch of computation accepts.

Theorem 2.2.5. *Let M be a* NDTM *and let $L = L(M)$. Then there exists a* TM *M' such that $L = L(M')$.*

Proof. By Theorem 2.2.2, it suffices to construct a multi-tape TM that simulates M. Construct a tree with branches corresponding to threads of computation given by M and nodes corresponding to configurations of M. We construct a 3-tape TM D as follows.

Tape 1 always contains the input string w and nothing else. Tape 2 contains just the string on the tape of the current node. Setting $m = \max\{|A| : A \in \operatorname{im} \delta\}$, tape 3 contains a string over $\{1, \ldots, m\}$ that corresponds to the "address" of the current node.

We make D simulate a breadth-first search of the tree as follows.

1. Initialize tape 1 with w and tape 3 with ϵ.
2. Copy tape 1 to tape 2.
3. Move to the node given by the next symbol on tape 3.
4. Read the configuration of M on w that is determined by this node.
5. Accept or reject if this is an accepting or rejecting configuration, respectively.
6. Replace the current string on tape 3 with the next string under the string order.
7. Do step 1.

□

Note 2.2.6.

1. This simulation has time complexity $O(m^T)$ where T denotes the steps taken by M.
2. We can modify the proof of our last theorem to show that if M always halts on each branch of computation, then M' always halts. Thus, a language is decidable if and only if a NDTM decides it.

Remark 2.2.7. There is an injective function ι from the set of Turning machines into $\{0,1\}^*$ because any TM's transition function admits a finite description. In particular, the set of TM is countable. Let $\langle M \rangle$ denote the binary encoding of the TM M. Let any $x \notin \operatorname{im} \iota$ correspond to the TM that immediately halts and outputs zero on every input. As a result, every binary string corresponds to some Turing machine.

Theorem 2.2.8. *There is a* TM *(denoted by U_{TM}) taking two strings as inputs, $\langle M \rangle$ and x, (i.e., one string over $\{0,1\} \times \Sigma$) such that*

1. *if M accepts x, then U_{TM} accepts,*
2. *if M rejects x, then U_{TM} rejects, and*
3. *if M does not halt on x, then neither does U_{TM}.*

We call U_{TM} a universal Turing machine. Moreover, if M takes T steps on w, then U_{TM} takes $O(T \log T)$ steps on $\langle M, w \rangle$.

Proof. This is like constructing an interpreter for a programming language within the language itself. See Arora and Barak, Theorem 1.13 for a high-level proof. □

2.3 Lecture 7

Note 2.3.1. Alternatively, we can view U_{TM} as a ternary function with input $(\langle M\rangle, w, 1^k)$ such that

1. if M accepts (resp. rejects) on w in $\leq k$ steps, then U_{TM} accepts (resp. rejects), and
2. if M does not halt on w in $\leq k$ steps, then U_{TM} will reach a special state.

Lemma 2.3.2. *Let A_{TM} denote the language $\{\langle M, w\rangle \mid M \text{ accepts } w\}$. Then A_{TM} is recursively enumerable.*

Proof. Observe that $L(U_{\mathsf{TM}}) = A_{\mathsf{TM}}$. □

Theorem 2.3.3. *A_{TM} is undecidable.*

Proof. Suppose, for contradiction, that there is some M that decides A_{TM}. Design a new TM N as follows.

(a) Given a binary string x, run M on $(\langle M_x\rangle, \langle M_x\rangle)$ where M_x denotes the TM corresponding to x.

(b) Let N reject when M accepts and N accept when M rejects.

Then

$$N(\langle N\rangle) = \begin{cases} \textit{accept} & N \text{ does not accept } \langle N\rangle \\ \textit{reject} & N \text{ accepts } \langle N\rangle \end{cases},$$

which is impossible. □

Lemma 2.3.4. *If L is decidable, then clearly $\overline{L}$ is decidable.*

Lemma 2.3.5. *If both L and $\overline{L}$ are recursively enumerable, then L is decidable.*

Proof. Find some M_1 and M_2 such that $L = L(M_1)$ and $\overline{L} = L(M_2)$. Construct some TM M as follows.

Algorithm 1: pseudocode describing M

```
Input: the string w
T = 1;
while the current state is a non-halting state do
    run U_TM on (⟨M_1⟩, w) for T steps;
    if U_TM accepts then
        accept
    else
        run U_TM on (⟨M_2⟩, w) for T steps ;
        if U_TM accepts then
            reject
        else
            T += 1
        end
    end
end
```

□

Corollary 2.3.6. *$\overline{A_{\mathsf{TM}}}$ is not recursively enumerable.*

Definition 2.3.7. A function $f : \Sigma^* \to \underline{\Sigma}^*$ is *computable* if there is some TM M such that for any string w, M halts on w with its tape containing just $f(w)$.

Definition 2.3.8. Let $L \subset \Sigma^*$ and $L' \subset \underline{\Sigma}^*$ be languages. We say that L *many-one reduces to* L', written as $L \leq_m L'$, if there is some computable function $f : \Sigma^* \to \underline{\Sigma}^*$ such that $x \in L \iff f(x) \in L'$. In this case, we call f the *reduction* from L to L'.

Lemma 2.3.9. *Suppose that $L \leq_m L'$ and that L' is decidable (resp. recursively enumerable), then L is decidable (resp. recursively enumerable). In this way, L' is at least as "hard" as L.*

Proof. Find some M that decides (resp. recognizes) L' and some reduction f from L to L'. Construct the TM N so that on input $w \in \Sigma^*$, we let N compute $f(w)$ and then output whatever M outputs on $f(w)$. Then N decides (resp. recognizes) L. □

2.4 Lecture 8

Example 2.4.1 (Halting problem). Let $A_{\text{HALT}} \coloneqq \{\langle M, w\rangle \mid M \text{ halts on } w\}$. Then A_{HALT} is undecidable.

Proof. Recall that A_{TM} is undecidable. Thus, it suffices to show that $A_{\mathsf{TM}} \leq_m A_{\text{HALT}}$. To do this, we want to design a computable function that maps any $\langle M, w\rangle$ to another $\langle M'\rangle, w'$ such that M' halts on w' precisely when M accepts w. Construct such an M' as follows.

Algorithm 2: pseudocode describing M'

Input: the string x
run U_{TM} on $(\langle M\rangle, x)$;
if U_{TM} *accepts* **then**
    accept
else
    while *true* **do**
        pass
    end
end

Then we get a suitable function given by $(\langle M\rangle, w) \mapsto (\langle M'\rangle, w)$. $\square$

Theorem 2.4.2 (Rice). *Every nontrivial semantic property of Turing machines is undecidable. Formally, let C be any subset of the universe of all languages over a fixed alphabet. Define $L_C = \{\langle M\rangle : L(M) \in C\}$. Suppose that both L_C and $\overline{L_C}$ are nonempty. Then L_C is undecidable.*

Proof. We may assume that $\emptyset \notin C$ for otherwise we could show that $\overline{L_C}$ is undecidable. We know that $L(M_y) \in C$ for some TM M_y. We show that $A_{\mathsf{TM}} \leq_m L_C$. Consider any $\langle M, w\rangle \in A_{\mathsf{TM}}$. Define M' as follows.

Algorithm 3: pseudocode describing M'

Input: the string x
run U_{TM} on $\langle M, w\rangle$;
if U_{TM} *accepts* **then**
    run U_{TM} on $\langle M_y, x\rangle$
else
    reject
end

If M accepts w, then $L(M') = L(M_y)$. If M rejects w, then $L(M') = \emptyset$. If M does not halt on w, then $L(M') = \emptyset$. $\square$

Example 2.4.3. Let $A_{\text{fin}} \coloneqq \{\langle M\rangle : L(M) \text{ is finite}\}$. Then A_{fin} is undecidable.

Example 2.4.4. Moreover, A_{fin} is not recursively enumerable.

Proof. Recall that $\overline{A_{\mathsf{TM}}}$ is not recursively enumerable. We show that $\overline{A_{\mathsf{TM}}} \leq_m A_{\text{fin}}$. Given any $\langle M, w\rangle$, define M' as follows.

Algorithm 4: pseudocode describing M'

Input: the string x
run U_{TM} on $\langle M, w\rangle$;
if U_{TM} *accepts* **then**
    accept
else
    reject
end

If M accepts w, then $L(M') = \{0,1\}^*$. Otherwise, $L(M') = \emptyset$. $\square$

Note 2.4.5. It's possible that both a language and its complement are not recursively enumerable.

3 Complexity theory

3.1 Lecture 9

Remark 3.1.1. Once we know that a question is decidable, we want to determine the amount of computational resources required to decide it. Such resources include

(a) time

(b) space

(c) parallelism

(d) communication

(e) rounds

(f) randomness .

We also want to study how these trade off with each other.

Definition 3.1.2. Given any TM M, its *time complexity* is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n) = \max\{\text{steps used by } M \text{ on input } w \mid |w| = n\}$. We say that M *runs in time* $f(n)$.

Definition 3.1.3. Given any *time-constructible* function $f : \mathbb{N} \to \mathbb{N}$, define $\mathsf{DTIME}(f(n)) = \{L : \exists \mathsf{TM}\ M$ such that $L = L(M)$ and M halts on all inputs of length n in $O(f(n))$ steps$\}$. Let $\mathbf{P} \coloneqq \bigcup_{k\geq 0} \mathsf{DTIME}(n^k)$.

Proposition 3.1.4. $\mathbf{P}$ *is independent of the variant of deterministic Turing machine used.*

Remark 3.1.5. Given any convex body, we want to compute its volume. In 1989, Dyer and Frieze proved that this is solvable in $O(n^{23})$ steps. It is now known that it's solvable in $O(n^2)$ steps.

Example 3.1.6. For any $k \geq 0$, $\mathsf{DTIME}(n^k) \subset \mathsf{DTIME}(n^{k+1})$. Consider the case where $k = 2$. Then we can show that this containment is proper by using diagonalization. Indeed, define the language L as follows.

1. If x is of the form $w10^i$ for some w and some i, then let $x \notin L$.

2. Otherwise, let M_w be the TM corresponding to w.

3. In this case, run M_w on x for n^2 steps where n denotes $|x|$.

4. If M_w does not halt in so many steps, then let $x \notin L$.

5. Else, let $x \in L$ when M_w rejects and let $x \notin L$ when M_w accepts.

Steps 2 and 3 together take $O(n^2 \log n)$ steps. It follows that $L \in \mathsf{DTIME}(n^2 \log n) \subset \mathsf{DTIME}(n^3)$. But $L \notin \mathsf{DTIME}(n^2)$.

Theorem 3.1.7 (Time hierarchy). *Let* $f : \mathbb{N} \to \mathbb{N}$ *be any function. Suppose that* $g(n) = \omega(f(n) \log f(n))$. *Then* $\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(g(n))$.

Definition 3.1.8. We say that a NDTM M has *time complexity* $t(n)$ if every branch of M runs in time $t(n)$. Define $\mathsf{NTIME}(t(n)) = \{L : \exists \mathsf{NDTM}\, M$ such that $L(M) = L$ and every branch of M halts on any input of length n in $O(t(n))$ steps$\}$. Let $\mathbf{NP} \coloneqq \bigcup_{k\geq 0} \mathsf{NTIME}(n^k)$.

Proposition 3.1.9. $\mathsf{NTIME}(t(n)) \subset \mathsf{DTIME}(2^{O(t(n))})$.

Example 3.1.10. Let φ be a Boolean formula in conjunctive normal form (CNF). Suppose that φ contains n clauses and m literals. Then the size of the representation in bits of φ is of order $O(2m \cdot n) = O(mn)$. Also, deciding whether φ evaluates to 1 takes $O(2m \cdot n) = O(mn)$ steps.

Define $\mathsf{CNF\text{-}SAT} = \{\langle \varphi \rangle : \varphi \text{ is satisfiable}\}$. Notice that this can be decided by a nondeterministic Turing machine in linear time. There is, however, no known deterministic Turing machine running faster than brute force.

3.2 Lecture 10

Lemma 3.2.1. *A language L belongs to* **NP** *if and only if there exist a deterministic* TM $V(\cdot,\cdot)$ *and constants $c_1, c_2 > 0$ such that $L = \{x \mid \exists y. |y| \leq |x|^{c_1} \wedge V(x,y) = 1$ where V runs in time $|x \cdot y|^{c_2}\}$. We call V a verifier and y a witness.*

Proof.
($\Longrightarrow$) There is some NDTM M running in polynomial time such that $L(M) = L$. Say that M runs in time n^c. The sequence of choices along any branch of M can be represented by a binary string of length n^c. Define V as the algorithm taking inputs x and y with $y \in \{0,1\}^{n^c}$ and executing M on x with choice of branch given by y. Then V runs in polynomial time, and $x \in L \iff V(x,y) = 1$ for some y.

($\Longleftarrow$) Given an input x, define a NDTM M that first guesses a witness y in a separate tape and then runs V on (x,y) in polynomial time. ☐

Note 3.2.2. Equivalently, we could have made V run in time $|x|^{c_2}$ while dropping the requirement that $|y|$ be polynomial in $|x|$.

Definition 3.2.3. An *independent set* of a graph $G = (V,E)$ is a set $I \subset V$ such that no two points in I are connected by an edge.

Example 3.2.4. The following languages are in **NP**.

1. $\mathsf{IND{-}SET} \coloneqq \{\langle G, k\rangle : G$ is an (undirected) graph with an independent set of size at least $k\}$
2. $\mathsf{3{-}COLOR} \coloneqq \{\langle G\rangle : G$ has a 3-coloring$\}$
3. $\mathsf{Composite} \coloneqq \{x \mid x$ is a composite number$\}$
4. $\mathsf{PRIMES} \coloneqq \{n \mid n$ is prime$\}$

Definition 3.2.5. We say that L_1 *polynomially many-one reduces to* L_2 (written as $L_1 \leq_m^p L_2$) if there is some TM M running in polynomial time such that $x \in L_1 \iff M(x) \in L_2$.

Lemma 3.2.6. *If $L_1 \leq_m^p L_2$ and $L_2 \in$ **P**, then $L_1 \in$ **P**.*

Definition 3.2.7. We say that L is **NP**-complete if $L \in$ **NP** and for any $L' \in$ **NP**, $L' \leq_m^p L$.

3.3 Lecture 11

Definition 3.3.1. A *(Boolean) circuit* is a directed acyclic graph with a unique sink node such that

1. each node has indegree at most 2
2. each internal node is labelled by $\wedge$, $\vee$, or $\neg$,
3. each leaf node is labeled by a Boolean variable, and
4. each edge is labeled by the Boolean value given as the output of the prior node.

The *size of a circuit* is the number of its internal nodes.

Remark 3.3.2. This is an example of a non-uniform model of computation as we must specify a new circuit for each input size.

Lemma 3.3.3. *Every function $g : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $O(2^n)$.*

Proof. We use induction to show that any $g : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $3 \cdot 2^n - 4$, which is enough. When $n = 1$, there are four cases to consider.

(a) If $g = \mathrm{id}_{\{0,1\}}$, then g can be computed by a circuit of size 0.

(b) If $g(0) = 1$ and $g(1) = 0$, then $g(x) = \neg x$.

(c) If $g(0) = g(1) = 0$, then $g(x) = x \wedge \neg x$.

(d) If $g(0) = g(1) = 1$, then $g(x) = x \vee \neg x$.

Hence the base case holds. Now, define $g_0, g_1 : \{0,1\}^{n-1} \to \{0,1\}$ by $g_0(y) = g(0, y)$ and $g_1(y) = g(1, y)$. Then g satisfies

$$g(y) = (\neg y_1 \wedge g_0(y_2, \dots, y_n)) \vee (y_1 \wedge g_1(y_2, \dots, y_n))$$

for each y. By induction, g can be computed by a circuit of size at most $4 + 2(3 \cdot 2^{n-1} - 4) = 3 \cdot 2^n - 4$. $\square$

Lemma 3.3.4. *Let $M = (Q, \Sigma, \Gamma, q_0, \delta, Q_F, Q_R)$ be a* TM*. Suppose that on any input of size n, M halts in at most t steps with $t \geq n$. Then there is a circuit of size $O(t^2 \cdot (|\Gamma| \cdot |Q|)^3) = O(t^2)$ that outputs 1 on a string x of length n if and only if M accepts x.*

Proof. We want to encode a given configuration of M, which we may assume uses at most t cells of tape. To do this, we take $\log |\Gamma|$ bits, 1 bit, and $\log |Q|$ bits to encode the content of the current cell, whether or not the head is located at this cell, and, if so, the current state, respectively. For each $i, j \geq 0$, the bit $b_{j,l+1}$ representing the j-th cell at time $l+1$ depends precisely on the three bits $b_{j-1,l}$, $b_{j,l}$, and $b_{j+1,l}$. If $B \coloneqq 1 + \log |\Gamma| + \log |Q|$, then every bit of the encoding of the configuration of M at time $l+1$ depends on $3B$ bits. This determines a Boolean function $f : \{0,1\}^{3B} \to \{0,1\}$ that computes the next configuration. Our previous lemma implies that f be be computed by a circuit of size $O(2^{3B})$ and hence by one of size $O((|\Gamma| \cdot |Q|)^3)$. Thus, there is a circuit of size $O(t \cdot t \cdot (|\Gamma| \cdot |Q|)^3)$ that simulates M on inputs of size n. $\square$

Example 3.3.5 (Cook-Levin theorem). Define $\mathsf{CIRCUIT{-}SAT} = \{\langle C \rangle : \exists x. C(x) = 1\}$. This is certainly in **NP**. We claim that it is **NP**-complete.

Proof. If $L \in \mathbf{NP}$, then there is an efficient (i.e., polynomial-time) algorithm V such that

$$\forall x \in L. \exists y \in \Sigma^*. |y| \leq |x|^{O(1)} \wedge V(x, y) = 1 \wedge (x \notin L \implies \forall y. V(x, y) = 0).$$

Thus, for each $n \in \mathbb{N}$, we can use our previous lemma to construct a circuit of size $n^{O(1)}$ such that $C(x, y) = V(x, y)$ for each string x of size n and each string y with $|y| \leq n^{O(1)}$. This means that for each string x, we can construct a circuit $C_x(\cdot)$ of size $|x|^{O(1)}$ such that $C_x(y) = V(x, y)$ for any y with $|y| \leq |x|^{O(1)}$. The mapping $M : x \mapsto \langle C_x(\cdot) \rangle$ satisfies $x \in L \iff M(x) \in \mathsf{CIRCUIT{-}SAT}$, as desired. $\square$

3.4 Lecture 12

Corollary 3.4.1. *Showing that* $\mathsf{CIRCUIT{-}SAT}$ *is not in* $\mathbf{P}$ *is equivalent to showing that* $\mathbf{P} \neq \mathbf{NP}$.

Corollary 3.4.2. *Suppose that* $\mathsf{CIRCUIT{-}SAT} \leq_m^p L$ *and* $L \in \mathbf{NP}$*. Then any* $L' \in \mathbf{NP}$ *satisfies* $L' \leq_m^p L$*, i.e.,* L *is* **NP***-complete.*

Note 3.4.3. Our last two corollaries hold with $\mathsf{CIRCUIT{-}SAT}$ replaced by any **NP**-complete language.

Definition 3.4.4. A Boolean formula in CNF is a *3cnf formula* if each clause contains exactly 3 literals.

Example 3.4.5.

1. Define $\mathsf{3{-}SAT} = \{\langle \varphi \rangle \mid \varphi \text{ is a 3cnf formula that is satisfiable}\}$. This is certainly in **NP**. We claim that it is **NP**-complete.

   Proof. It suffices to show that $\mathsf{CIRCUIT{-}SAT} \leq_m^p \mathsf{3{-}SAT}$. We must construct an efficient algorithm $M(-)$ such that the circuit C is satisfiable if and only if $\varphi \coloneqq M(\langle C \rangle)$ is satisfiable. If C has size n, then, wlog, we can use the associativity of our Boolean operations to add at most n^k internal nodes to C such that each gate labeled by $\wedge$ or $\vee$ takes exactly two inputs.

   Let $g_1, \dots, g_n$ and $x_1, \dots, x_m$ denote the Boolean values given by the edges and inputs of C, respectively. Relabel $g_1, \dots, g_n, x_1, \dots, x_m$ as $w_1, \dots, w_{n+m}$. Let φ be the 3cnf formula in the variables $w_1, \dots, w_{n+m}$ where each clause of φ corresponds either to C's output value $w_s \vee w_s \vee w_s$ or to one of C's internal edges. In the latter case, we can give the following descriptions.

- If $w_j = \neg w_i$, then φ contains exactly one clause of the form

$$(w_i \vee w_j) \wedge (\neg w_i \vee \neg w_j).$$

- If $w_h = w_i \wedge w_j$ in C, then φ contains exactly one clause of the form

$$(w_i \vee w_j \vee \neg w_h) \wedge (w_i \vee \neg w_j \vee \neg w_h) \wedge (\neg w_i \vee w_j \vee \neg w_h) \wedge (\neg w_i \vee \neg w_j \vee w_h).$$

- If $w_h = w_i \vee w_j$ in C, then φ contains exactly one clause of the form

$$(w_i \vee w_j \vee \neg w_h) \wedge (w_i \vee \neg w_j \vee w_h) \wedge (\neg w_i \vee w_j \vee w_h) \wedge (\neg w_i \vee \neg w_j \vee w_h).$$

By construction, φ is satisfiable if and only if C is. The algorithm $M : \langle C \rangle \mapsto \varphi$ is linear in n^k, hence efficient. Hence it is a suitable reduction. $\square$

2. It's clear that $\mathsf{IND{-}SET}$ is in **NP**. We claim that this is **NP**-complete.

Proof. We show that $\mathsf{3{-}SAT} \leq_m^p \mathsf{IND{-}SET}$. Let φ be a 3cnf-formula and write $\varphi = c_1 \wedge c_2 \wedge c_3 \wedge \cdots \wedge c_m$. For each clause c_i, create a triangle t_i with vertices corresponding to the three literals in c_i. Let G_φ denote the graph obtained from the graph $\coprod_{i=1}^m t_i$ by adding an edge between any two conflicting vertices v and $\neg v$ in $\coprod_{i=1}^m t_i$. Then the algorithm $\varphi \mapsto \langle G_\varphi, m \rangle$ defines a suitable reduction. $\square$

3. We say that $K \subset V$ is a *vertex cover* of a graph $G = (V, E)$ if any $(x, y) \in E$ has $x \in K$ or $y \in K$. Define $\mathsf{VC} = \{\langle G, k \rangle \mid G \text{ has a vertex cover of size at most } k\}$. Then $\mathsf{IND{-}SET} \leq_m^p \mathsf{VC}$, so that VC is **NP**-complete.

Proof. Let G be a graph with an independent set S with $|S| \geq k$. Then $V \setminus S$ is a vertex cover of G. Conversely, if G has a vertex cover K of size at most $|V| - k$, then $V \setminus K$ is an independent set of size at least k in G. Thus, the algorithm $\langle G, k \rangle \mapsto \langle G, |V| - k \rangle$ defines a suitable reduction. $\square$

3.5 Lecture 13

Definition 3.5.1. We say that a language L is **NP**-*hard* if $L' \leq_m^p L$ for any $L' \in \mathbf{NP}$.

Definition 3.5.2. Let $G = (V, E)$ be a graph. A subset $C \subset V$ is a *clique in G* if any two distinct points in C are adjacent. Let $\mathsf{CLIQUE} \coloneqq \{\langle G, k \rangle \mid G \text{ has a clique of size at least } k\}$.

Definition 3.5.3. If $G = (V, E)$ is a graph, then the graph $\overline{G} = (V, E')$ where $E' = \{(x, y) \mid (x, y) \notin E\}$ is the *complement graph of G*

Proposition 3.5.4. *A set S of vertices in a graph G is an independent set in G if and only if it is a clique in $\overline{G}$.*

Example 3.5.5. $\mathsf{IND{-}SET} \leq_m^p \mathsf{CLIQUE}$ via $\langle G, k \rangle \mapsto \langle \overline{G}, k \rangle$. Hence CLIQUE is **NP**-hard.

Definition 3.5.6. Let $G = (V, E)$ be an (undirected) weighted graph with weight function $w : E \to \mathbb{R}_{\geq 0}$. Suppose $C \subset V$. A *Steiner tree in G* is a connected subgraph of G with no cycles that contains each vertex in C.

Remark 3.5.7. Any connected subgraph of minimum weight must be a tree.

Problem. Given a weighted graph G and set of vertices C in G, find the Steiner tree in G of minimum weight that contains C.

Definition 3.5.8. Let $G = (V, E)$. A subset $C \subset V$ *has a Steiner tree of total weight W* if there exists a connected subgraph of G that contains every vertex in C and has weight at most W.

Example 3.5.9. Let $\mathsf{Steiner{-}tree} \coloneqq \{\langle G, C, W \rangle \mid C \text{ has a Steiner tree of total weight at most } W\}$. This is **NP**-complete.

Proof. It's clear that $\mathsf{Steiner{-}tree}$ is in $\mathbf{NP}$. To show that it is also $\mathbf{NP}$-hard, we prove that $\mathsf{VC} \leq_m^p \mathsf{Steiner{-}tree}$. Let $G = (V, E)$ be a graph with a vertex cover S of size k.

Let
$$V' = \bigcup_{v \in V} [v] \bigcup_{(u,v) \in E} [u, v].$$

Build E' as follows.

(a) Let $([u], [v]) \in E'$ for any $u, v \in V$ and set $w'([u], [v]) = 1$.

(b) If $(u, v) \in E$, then let $([u], [u, v]), ([u, v], [v]) \in E'$ and set $w'([u], [u, v]) = w'([u, v], [v]) = 1$.

(c) If $(v, w) \in E$ and u, v, w are pairwise distinct, then let $([u], [v, w]) \in E'$ and and set $w'([u], [v, w]) = 2$.

(d) Finally, for any $(u, v), (w, z) \in E$, let $([u, v], [w, z]) \in E'$ with weight
$$w'([u, v], [w, z]) = \begin{cases} 2 & (u, v) \text{ and } (w, z) \text{ share a vertex} \\ 3 & \text{otherwise} \end{cases}.$$

Set $C = \{[u, v] : (u, v) \in E\}$. Also, set $W = |E| + k - 1$. Let $G' = (V', E', w')$. Note that we have constructed G' in polynomial time.

Claim. *G' has a Steiner tree containing C with weight at most $|E| + k - 1$.*

Proof. Write $S = \{v_1, \ldots, v_k\}$. Let $S' \coloneqq \{[v_1], \ldots, [v_k], \bigcup_{(u,v) \in E} [u, v]\}$ and $E_{S'} \coloneqq \{(x, y) \in E' \mid w'(x, y) = 1$ and $x, y \in S'\}$. Since S is a vertex cover, we see that $(S', E_{S'})$ is a connected subgraph of G' that contains C and has weight $|E| + k - 1$. □

Claim. *If C has a a Steiner tree T of total weight $W \leq |E| + k - 1$, then G has a vertex cover of size k.*

Proof. Alter T as follows.

- Replace any edge of weight 2 between $[w]$ and $[u, v]$ with the edges $([w], [u])$ and $([u], [u, v])$.
- Replace any edge of weight 2 between $[u, v]$ and $[v, w]$ with the edges $([u, v], [v])$ and $([v], [v, w])$.
- Replace any edge of weight 3 between $[u, v]$ and $[w, z]$ with the edges $([u, v], [v])$, $([v], [w])$, and $([w], [w, z])$.

The resultant graph T' is connected and contains C. Note that T' has weight at most $|E| + k - 1$ where each edge of T' has weight 1. This implies that T' spans at most $|E| + k$ vertices. Since T' contains C, it follows that T' contains a set R of vertices of the form $[v]$ such that $|R| \leq k$. If $(x, y) \in E$, then $[x, y]$ is connected to some $[v_0]$ by edges of weight 1. This means that $[x]$ or $[y]$ is in T', so that $[x]$ or $[y]$ is in R. This shows that R is a vertex cover for G. □

□

Remark 3.5.10. Any undirected weighted (connected) graph can be endowed with a metric by taking the shortest path between any two vertices. Our choice of weights in part (d) of our construction of E' made G' a metric space.

Example 3.5.11. Let $\mathsf{SUBSET{-}SUM} \coloneqq \{\langle a_1, \ldots, a_k, t\rangle \mid a_i, t \geq 0,\ \exists S \subset [k].\, \sum_{i \in S} a_i = t\}$. $\mathsf{SUBSET{-}SUM}$ is $\mathbf{NP}$-complete.

Proof. It's clear that this is in **NP**. We show that $\mathsf{VC} \leq_m^p \mathsf{SUBSET{-}SUM}$. Let $G = (V, E)$ such that $|V| = n$ and $|E| = m$. We can make E totally ordered. Suppose that G has a vertex cover C of size k. For each $v \in V$, we define an integer $a_v \geq 0$ in base-4 (written from left to right) consisting of $m+1$ digits. Further, for each $e \in E$, we define an integer $b_e \geq 0$ in base-4 consisting of $m+1$ digits. Specifically, if $0 \leq i \leq |E|-1$ and $(u, v) \in E$ is the i-th edge, then define both a_u and a_v as the integer

$$0 \cdots 0 \underbrace{1}_{i\text{-th digit}} 0 \cdots 01$$

and define $b_{(u,v)}$ as the integer

$$0 \cdots 0 \underbrace{1}_{i\text{-th digit}} 0 \cdots 00.$$

Now, set $t = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$.

Let $S = \{a_v \mid v \in C\} \cup \{b_{(u,v)} \mid \text{exactly one of } u \text{ and } v \text{ belongs to } C\}$. Note that we can construct S in polynomial time. It's straightforward to check that the terms of S sum to t.

Claim. *Suppose that there are $U \subset V$ and $T \subset E$ such that $t = \sum_{u \in U} a_u + \sum_{(u,v) \in T} b_{(u,v)}$. Then U is a vertex cover for G of size at most k.*

Proof. Since $t < (k+1)4^m$ and each $a_u > 4^m$, it follows that $|U| \leq k$. Note that, in base-4, each of the first m digits of t equals 2. Thus, for each $(u, v) \in E$, at least two of a_u, a_v, and $b_{(u,v)}$ contribute to the summation $\sum_{u \in U} a_u + \sum_{(u,v) \in T} b_{(u,v)}$. This implies that at least one of u and v belongs to U. Hence U is a vertex cover for G. □

□

Remark 3.5.12. Using dynamic programming, one can show that there is a $\mathrm{poly}(k, t)$ algorithm deciding $\mathsf{SUBSET{-}SUM}$. This result, however, does not imply that $\mathsf{SUBSET{-}SUM} \in \mathbf{P}$, because the size of the whole input $\langle a_1, \ldots, a_k, t \rangle$ is on the order of $k \log t$.

3.6 Lecture 14

Definition 3.6.1. Let $S : \mathbb{N} \to \mathbb{N}$.

1. Define the *space complexity class* $\mathsf{DSPACE}(S(n)) = \{L \mid \exists\, \mathsf{TM}\, M$ such that $L = L(M)$ and on any input of length n, M touches at most $S(n)$ cells (on its work tape)$\}$.
2. Define $\mathsf{NSPACE}(S(n)) = \{L \mid \exists\, \mathsf{NDTM}\, M$ such that $L = L(M)$ and on any input of length n, M halts on every branch of computation and touches at most $S(n)$ cells on any branch$\}$.

Unless we state otherwise, we assume that $S(n) \geq \log n$.

Definition 3.6.2. Let M be a Turing machine that always halts. Define the *configuration graph $G_{M,x}$ of M on input x* as the directed graph (V, E) where V consists of the possible configurations of M on x and $E = \{(C, C') : C \vdash C'\}$. By making M erase the contents of its work tapes right before halting, we may assume that M has exactly one accepting configuration C_{accept} on x.

Note 3.6.3. Since M always halts, it can never reach the same configuration more than once. Thus, $G_{M,x}$ is a directed acyclic graph.

Lemma 3.6.4. $\mathsf{NSPACE}(S(n)) \subset \mathsf{DTIME}(2^{O(S(n))})$.

Proof. Let $L \in \mathsf{NSPACE}(S(n))$ with $L(M) = L$. Given any input x of length n, we can use $O(S(n))$ bits to describe the current contents of the tape, $O(\log S(n))$ bits to describe the current state, and $O(1)$ bits to describe the current location of the head. Thus, we need $O(S(n)) + O(1) + O(\log S(n)) = O(S(n))$ bits to describe any vertex of $G_{M,x}$.

Note that the number of configurations of M is at most $2^{O(S(n))}$ (provided that any configuration of a NDTM yields at most two distinct configurations). Therefore, we can construct $G_{M,x}$ in $2^{O(S(n))}$ steps. Now apply the standard linear-time BFS for connectivity to $G_{M,x}$ to decide if there is a path from C_{start} to C_{accept}. $\square$

Corollary 3.6.5.

1. $\mathsf{DTIME}(S(n)) \subset \mathsf{DSPACE}(S(n)) \subset \mathsf{NSPACE}(S(n)) \subset \mathsf{DTIME}(2^{O(S(n))})$.
2. $\mathsf{DTIME}(S(n)) \subset \mathsf{NTIME}(S(n)) \subset \mathsf{NSPACE}(S(n))$.

Remark 3.6.6. It is not known whether these chains of containment can be improved.

Definition 3.6.7.

1. Define $\mathsf{PSPACE} = \bigcup_{k\geq 0} \mathsf{DSPACE}(n^k)$.
2. Define $\mathsf{NPSPACE} = \bigcup_{k\geq 0} \mathsf{NSPACE}(n^k)$.

Note 3.6.8.

1. $\mathbf{P} \subset \mathsf{PSPACE}$.
2. $\mathbf{NP} \subset \mathsf{NPSPACE}$.

Proposition 3.6.9. *Let* $\mathbf{L} := \mathsf{DSPACE}(\log n)$ *and* $\mathbf{NL} := \mathsf{NSPACE}(\log n)$.

1. *Let* $L_1 = \{\langle x, y, z\rangle \mid x \cdot y = z\}$ *and* $L_2 = \{\langle x, y, z\rangle \mid x + y = z\}$. *Then* $L_1, L_2 \in \mathbf{L}$.
2. *Let* $\mathsf{DIR{-}REACH} := \{\langle G, s, t\rangle \mid G$ *is directed and the vertex* t *is reachable from* $s\}$. *Then* $\mathsf{DIR{-}REACH} \in \mathbf{NL}$.

   Remark 3.6.10. Omer Reingold has shown that $\mathsf{REACH} := \{\langle G, s, t\rangle \mid G$ is undirected and the vertex t is reachable from $s\}$ belongs to $\mathbf{L}$.

Theorem 3.6.11 (Savitch). *Recall that* $\mathsf{NTIME}(S(n)) \subset \mathsf{DTIME}(2^{O(S(n))})$. *But* $\mathsf{NSPACE}(S(n)) \subset \mathsf{DSPACE}(S^2(n))$.

Corollary 3.6.12. $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

3.7 Lecture 15

Theorem 3.7.1 (Savitch). *Recall that* $\mathsf{NTIME}(S(n)) \subset \mathsf{DTIME}(2^{O(S(n))})$. *But we have that*

$$\mathsf{NSPACE}(S(n)) \subset \mathsf{DSPACE}(S^2(n)).$$

Proof. Let $L \in \mathsf{NSPACE}(S(n))$ with $L(M) = L$. Recall that the configuration graph $G_{M,x}$ has at most $T_0 := 2^{O(S(n))}$ nodes. Consider the following recursive algorithm.

Input: the string x
for $j \in \{1, \ldots, T_0\}$ **do**
  if $\mathsf{REACH}(C_{start}, j, \frac{T_0}{2})$ *and* $\mathsf{REACH}(j, C_{accept}, \frac{T_0}{2})$ **then**
    output "yes"
  else
    output "no"
  end
end

Denote the space complexity of the preceding algorithm by $\mathcal{L}(T_0)$. Note that we we can reuse the space used by the first recursive call for the second recursive call. Since we need $\log T_0$ cells to encode the counter, it follows that

$$\mathcal{L}(T_0) = \log T_0 + \mathcal{L}(T_0/2) + O(1).$$

Note that the recursion depth is precisely $\log T_0$. Using this, we compute

$$\begin{aligned}\mathcal{L}(T_0) &= \log T_0 + \mathcal{L}(T_0/2)\\ &= \log^2 T_0 + \mathcal{L}(1) = \log^2 2^{O(S(n))} + \mathcal{L}(1)\\ &= O(S^2(n)) + O(S(n)) = O(S^2(n)).\end{aligned}$$

□

Corollary 3.7.2. **NL** $\subset$ **P** *because* DIR−REACH *is* **NL**-*complete and, by our last proof, has a* $\log^2 n$-*space deterministic algorithm.*

Note 3.7.3.

1. We have that $\mathsf{NTIME}(\text{poly}(n)) \subset \mathsf{NPSACE}(\text{poly}(n)) \subset \mathsf{DPSACE}(\text{poly}(n)) = \mathsf{PSPACE}$.
2. PSPACE is closed under complementation.

Example 3.7.4.

1. Let $\Sigma_2\text{−}\mathsf{SAT} \coloneqq \{\varphi \text{ Boolean} \mid \forall \bar{x} \exists \bar{y} (\varphi(\bar{x}, \bar{y}) = 1)\}$. It is unclear that this (or its complement) belongs to **NP**.
2. Let $\mathsf{TQBF}\text{−}\mathsf{SAT} \coloneqq \{\varphi(\overline{x_1}, \ldots, \overline{x_n}) \mid Q_1\overline{x_1}Q_2\overline{x_2}Q_3\overline{x_3}\cdots Q_n\overline{x_n}(\varphi(\overline{x_1}, \ldots, \overline{x_n}) = 1),\ Q_i \in \{\forall, \exists\}\}$. This stands for the set of *totally quantified Boolean formulas.* It belongs to PSPACE.

   Proof. Construct an algorithm $T(\varphi)$ as follows.

   - If φ is quantifier-free, then evaluate it directly. Accept if it evaluates to 1 and reject otherwise.
   - If $\varphi = \exists x \psi$, then recursively call T on ψ once with $x = 0$ and once with $x = 1$. Accept if either of these recursive calls accepts and reject otherwise.
   - If $\varphi = \forall x \psi$, then recursively call T on ψ once with $x = 0$ and once with $x = 1$. Accept if both of these recursive calls accept and reject otherwise.

   If m denotes the size of φ, then $\mathcal{L}(m) = m^{O(1)} + \mathcal{L}(m-1) + O(1) = m^{O(1)} + \mathcal{L}(m-1) = O(m^k)$ for some k. □

3.8 Lecture 16

Definition 3.8.1. A language L is PSPACE-complete if it belongs to PSPACE and for any $L' \in \mathsf{PSPACE}$, $L' \leq_m^p L$.

Example 3.8.2. TQBF−SAT is PSPACE-complete.

Proof. Let $L \in \mathsf{PSPACE}$ with $L(M) = L$. Given any input x with $|x| = n$, we want to construct a TQBF $\varphi_{c,c',i}$ of size $O(S(n)^2)$ that is is satisfiable if and only if there is a path of length at most 2^i from c to c' in the configuration graph $G_{M,x}$. This will imply that

$$\hat{\varphi} \coloneqq \varphi_{C_{\text{start}}, C_{\text{accept}}, O(S(n))}$$

is true if and only if M accepts x if and only if $x \in L$.

We have previously constructed such a $\varphi_{c_1,c_2,0}$. Moreover, if $i \geq 1$, then we see that

$$\begin{aligned}\varphi_{c_1,c_2,i} &\equiv \exists c(\varphi_{c_1,c,i-1} \wedge \varphi_{c,c_2,i-1})\\ &\equiv \exists c \forall D^1 \forall D^2((D^1 = c_1 \wedge D^2 = c) \vee (D^1 = c) \wedge (D^2 = c_2)) \implies \varphi_{D^1,D^2,i-1}.\end{aligned}$$

It follows that $|\varphi_{c_1,c_2,i}| \leq |\varphi_{c_1,c_2,i-1}| + O(S(n))$, so that $|\hat{\varphi}| \leq O(S(n)^2)$, as desired. □

Proposition 3.8.3. *A language L belongs to* **NL** *if and only if there exist $c \in \mathbb{N}$ and a deterministic* TM *$V(\cdot,\cdot)$ consisting of one read-only input tape, one work tape, and one read-only, single-axis proof tape such that V's work tape uses $O(\log|(\textit{first input})|)$ space and*

$$L = \{x \mid \exists y. |y| \leq |x|^c \wedge V(x,y) = 1\}.$$

.

3.9 Lecture 17

Definition 3.9.1.

1. Let M be a TM consisting of one read-only input tape, one work tape, and one write-only, single-axis output tape such that M's work tape uses $O(\log n)$ space. We call M a *log space transducer.*
2. Let A and B be languages. We say that A is *log space reducible to* B, written as $A \leq_l B$, if there is some log space transducer $M : \Sigma^* \to \underline{\Sigma}^*$ such that $x \in A \iff M(x) \in B$.
3. A language L is **NL**-*complete* if it is in **NL** and $L' \leq_l L$ for any $L' \in \mathbf{NL}$.

Proposition 3.9.2. *If* $L \in \mathbf{NL}$ *and* $L' \leq_l L$*, then* $L' \in \mathbf{NL}$.

Example 3.9.3. $\mathsf{DIR{-}REACH}$ is **NL**-complete.

Proof. See Arora and Barak, Theorem 4.16. □

Theorem 3.9.4 (Immerman-Szelepcsényi).

$$\mathbf{NL} = \mathbf{co{-}NL}.$$

Proof. Since $\mathsf{DIR{-}REACH}$ is **NL**-complete, it suffices to show that $\mathsf{DIR{-}REACH}^c \in \mathbf{NL}$. Let $G = (V, E)$ be a graph of size n and $s, t \in V$. Let C_i denote the set of vertices $v \in V$ reachable from s in at most i steps. We can assume that V is ordered $(v_1, \ldots, v_n)$ since each index can be described in $\log n$ bits.

First, let $v \in V$. Given that $|C_i| = k$, we can verify that $v \notin C_i$ as follows.

1. Propose a list $v_{s_1}, \ldots, v_{s_m}$ of vertices and a path $s \rightsquigarrow v_{s_i}$ for each $1 \leq i \leq m$.
2. Write the next v_{s_i} on the work tape and write the proposed path $s \rightsquigarrow v_{s_i}$ on the proof tape.
3. Verify that the proposed path is valid. If not, then *reject.*
4. Otherwise, verify that $s_i > s_{i-1}$ and $v_{s_i} \neq v$. If not, then *reject.*
5. Repeat steps 2-4 until there is no v_{s_i} left.
6. Verify that $m = k$ by using a counter on the work tape. If not, then *reject.* Otherwise, *accept.*

Second, given that $|C_{i-1}| = k$, we can verify that $v \notin C_i$ as follows.

1. Propose a list $v_{s_1}, \ldots, v_{s_m}$ of vertices and a path $s \rightsquigarrow v_{s_i}$ for each $1 \leq i \leq m$.
2. Write the next v_{s_i} on the work tape and write the proposed path $s \rightsquigarrow v_{s_i}$ on the proof tape.
3. Verify that the proposed path is valid. If not, then *reject.*
4. Otherwise, verify that $s_i > s_{i-1}$, $v_{s_i} \neq v$, and $\underline{v_{s_i} \text{ is not a neighbor of } v}$. If not, then *reject.*
5. Repeat steps 2-4 until there is no v_{s_i} left.
6. Verify that $m = k$ by using a counter on the work tape. If not, then *reject.* Otherwise, *accept.*

Finally, let $c \in \mathbb{N}$. Given that $|C_{i-1}| = k$, we can verify that $|C_i| = c$ as follows.

1. Write the next v_i on the work tape (where $1 \leq i \leq n$).
2. Decide if $v_i \in C_i$ using our two previous algorithms.
3. Repeat steps 1-2 until there is no v_i left.
4. Determine the number r of vertices in C_i by using a counter on the work tape. If $r = c$, then *accept.* Otherwise, *reject.*

Note that each of our three verifiers uses $O(\log n)$ space on its work tape and is polynomial in n on its proof tape. Apply our final algorithm iteratively n-times to verify the size of C_n. Since we can reuse space on the work tape, our space complexity on it remains $O(\log n)$. Next, apply our first algorithm to verify that $t \notin C_n$, in which case t is not reachable from s. $\square$

Corollary 3.9.5. *If $s(n) \geq \log n$, then* $\mathsf{NSPACE}(s(n)) = \mathbf{co}\text{−}\mathsf{NSPACE}(s(n))$.

Remark 3.9.6. Bertrand's postulate implies that some prime number between N and $2N$ always exists. Suppose that we want to find the least such prime $\tilde{p}$. Consider the probability $\mathbb{P}$ that a randomly chosen number between N and $2N$ is prime. From the prime number theorem, it is known that $\mathbb{P} \approx \frac{N}{\log N}$. As a result, we can apply the AKS primality test $O(\log N)$ times to find $\tilde{p}$ with high probability.

3.10 Lecture 18

Definition 3.10.1.

1. We call a TM a *probabilistic/randomized Turing machine* if it consists of an input tape, a work tape, and a "random bits" tape.

2. Let $p : \mathbb{N} \to \mathbb{N}$ be a polynomial. A probabilistic TM $M(\cdot, \cdot)$ *decides L with respect to p* if

   - for any $x \in L$, $\mathbb{P}_{r \in_R \{0,1\}^{p(|x|)}}[M(x, r) = 1] \geq \frac{2}{3}$ and
   - for any $x \notin L$, $\mathbb{P}_{r \in_R \{0,1\}^{p(|x|)}}[M(x, r) = 0] \geq \frac{2}{3}$.

Definition 3.10.2. A language L is *in* $\underbrace{\mathsf{BPTIME}(t(n))}_{\textit{bounded error probabilistic time}}$ if there exists a randomized TM M running in time $t(|x|)$ (with probability 1) such that M decides L with respect to $t(n)$. Let $\mathbf{BPP} \coloneqq \bigcup_{c \geq 0} \mathsf{BPTIME}(n^c)$.

Note 3.10.3. It's clear that $\mathbf{P} \subset \mathbf{BPP}$.

Proposition 3.10.4. $\mathbf{P} \neq \mathbf{NP} \implies \mathbf{P} = \mathbf{BPP}$.

Remark 3.10.5.

1. Computing the value of the determinant of an $n \times n$ matrix of integers via cofactor expansion takes $\omega(n!)$ steps. Computing it via Gaussian elimination, however, takes $O(n^3)$ steps.

2. Computing the value of the determinant of an $n \times n$ matrix of linear forms over $\mathbb{Z}$ via cofactor expansion is exponential in n. There is no known deterministic polynomial time algorithm for such a computation.

Proposition 3.10.6.

(a) Let L be an $n \times n$ matrix of linear forms in $\mathbb{Z}[x_1, \ldots, x_n]$ whose coefficients are in $[-2^n, 2^n]$. Then $\det L$ *is a polynomial in $x_1, \ldots, x_n$ with (total) degree n and each coefficient an integer $\leq 2^{O(n^2)}$.*

(b) Let $p(x)$ be a univariate polynomial of degree $d \geq 0$. Let $S \subset \mathbb{Z}$ be finite. Then $\mathbb{P}[p(x) = 0] \leq \frac{d}{|S|}$ for any $x \in_R S$.

Lemma 3.10.7 (DeMillo-Lipton-Schwartz-Zippel). *Let $p(x_1, \ldots, x_n)$ be a multivariate polynomial of degree at most $d \geq 0$. Let $S \subset \mathbb{Z}$ be finite. Then for any $a_1, \ldots, a_n$ randomly chosen with replacement from S,*

$$\mathbb{P}[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}.$$

Proof. We use induction on n. If $n = 1$, then this is exactly Proposition 3.10.6(b). Now, we can write

$$p(x_1, \ldots, x_n) = \sum_{i=0}^{d} = x_1^i q_i(x_2, \ldots, x_n)$$

where $\deg q_i \le d - i$ for each i. Let k be maximal such that $q_k(x_2, \ldots, x_n) \neq 0$. Let E denote the event that $q_k(x_2, \ldots, x_n) = 0$. By induction together with Proposition 3.10.6(b), it follows that

$$\begin{aligned}\mathbb{P}[p(a_1, \ldots, a_n) = 0] &\le \mathbb{P}[E] + \mathbb{P}[p(a_1, \ldots, a_n) = 0 \mid \neg E] \\ &\le \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}.\end{aligned}$$

□

Example 3.10.8. Let L be an $n \times n$ matrix of linear forms in $\mathbb{Z}[x_1, \ldots, x_n]$ whose coefficients are in $[-2^n, 2^n]$. Define the probabilistic TM A on input $\langle L \rangle$ as follows.

1. Set $S = \{1, \ldots, 100n\}$.
2. Choose $a_1, \ldots, a_n$ randomly from S with replacement
3. Evaluate $\det(a_1, \ldots, a_n)$. If this equals 0, then *accept*. Otherwise, *reject*.

Then A accepts $\langle L \rangle$ with probability 1 when $\det L = 0$. Also, it rejects with probability $\ge \frac{99}{100}$ when $\det L \neq 0$ because

$$\mathbb{P}[\det(a_1, \ldots, a_n) = 0] \le \frac{1}{100}.$$

Since evaluating a polynomial is polynomial in its degree, we see that A is polynomial in n.

3.11 Lecture 19

Example 3.11.1. Let $G = (V_1, V_2, E)$ be a bipartite graph with $|V_1 = |V_2| = n$. A *perfect matching* is a permutation σ of $\{1, 2, \ldots, n\}$ such that $(i, \sigma(i)) \in E$ for each $i = 1, \ldots, n$.

Let $M = (m_{i,j})$ be the $n \times n$ matrix with

$$m_{i,j} = \begin{cases} X_{ij} & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}.$$

Since

$$\det M = \sum_{\sigma \in S_n} (-1)^{\operatorname{sgn} \sigma} \prod_{i=1}^{n} m_{i, \sigma i},$$

we see that $\det M \neq 0$ if and only if G has some perfect matching. By Example 3.10.8, it follows that deciding whether or not a finite graph has a perfect matching is in **BPP**.

Lemma 3.11.2 (Chernoff bound). *Let $X_1, \ldots, X_n$ be independent boolean-valued random variables. Let $\mathbb{E}[X_i] = \mu$. Then $Z = \frac{X_1 + \cdots + X_n}{n}$, so that $\mathbb{E}[Z] = \mu$. Then*

$$\mathbb{P}[|Z - \mu| \ge t] \le e^{\frac{-t^2 n \mu}{4}}$$

for any $t \in (0, 1)$.

Corollary 3.11.3. *Let M be a randomized polynomial-time TM and $L \subset \Sigma^*$ be a language. Suppose that there exists $c > 0$ such that*

- *for any $x \in L$, $\mathbb{P}_{r \in_R \{0,1\}^{\mathrm{poly}(|x|)}}[M(x, r) = 1] \ge \frac{1}{2} + n^{-c}$ and*
- *for any $x \notin L$, $\mathbb{P}_{r \in_R \{0,1\}^{\mathrm{poly}(|x|)}}[M(x, r) = 0] \ge \frac{1}{2} + n^{-c}$*

where n denotes $|x|$. Then for any $c' > 0$, there exist a randomized polynomial-time TM M' such that

- *for any $x \in L$, $\mathbb{P}_{r \in_R \{0,1\}^{\mathrm{poly}(|x|)}}[M'(x, r) = 1] \ge 1 - 2^{-n^{c'}}$ and*
- *for any $x \notin L$, $\mathbb{P}_{r \in_R \{0,1\}^{\mathrm{poly}(|x|)}}[M'(x, r) = 0] \ge 1 - 2^{-n^{c'}}$.*

Proof. Set $m = n^{2c+2c'+100}$. Define M' as follows. On any input x, run $M(x)$ m times, with outputs $y_1, \ldots, y_m$. Accept if M accepts x more than $\frac{m}{2}$ times and reject otherwise.

For each $i \in \{1, \ldots, m\}$, define the random variable

$$X_i = \begin{cases} 1 & y_i = \chi_L(x) \\ 0 & \text{otherwise} \end{cases}.$$

Then $\mathbb{E}[X_i] = \mathbb{P}[X_i = 1] \geq \mu \coloneqq \frac{1}{2} + n^{-c}$. We can apply the Chernoff bound to get

$$\begin{aligned}
\mathbb{P}[\sum_{i=1}^{m} X_i \leq \frac{m}{2}] &\leq \mathbb{P}[\Big|\frac{\sum_{i=1}^{m} X_i}{m} - \mu\Big| \geq n^{-c}] \\
&\leq e^{\frac{-n^{-2c}(n^{2c+2c'+100})(\frac{1}{2}+n^{-c})}{4}} \\
&= \frac{1}{e^{\frac{n^{2c'+100}(\frac{1}{2}+n^{-c})}{4}}} \\
&= \frac{1}{e^{\frac{n^{2c'+100}}{8} + \frac{n^{2c'-c+100}}{4}}} \\
&= \frac{1}{e^{\frac{n^{2c'+100} + 2n^{2c'-c+100}}{8}}} \\
&\leq \frac{1}{e^{\frac{1}{8}n^{2c'+100}}} \leq \frac{1}{2^{n^{c'}}}.
\end{aligned}$$

$\square$

Remark 3.11.4. Call a random bit r *bad for x* if $M(x,r) \neq \chi_L(x)$ and *good for x* otherwise. For any x of length n, $\mathbb{P}_{r \in_R \{0,1\}^{\text{poly}(|x|)}}[r \text{ is bad for } x] \leq 2^{-n^c}$. Thus, $\mathbb{P}_{r \in_R \{0,1\}^{\text{poly}(|x|)}}[r$ is bad for some x of length $n] \leq 2^{-n^c} \cdot 2^n \ll 2^{-n}$ when c is large, and $\mathbb{P}_{r \in_R \{0,1\}^{\text{poly}(|x|)}}[r$ is good for every x of length $n] \geq 1 - 2^{-n}$.

Definition 3.11.5.

1. Let **RP** consist of those languages L for which there is some efficient randomized TM M such that
   - for any $x \in L$, $\mathbb{P}_{r \in_R \{0,1\}^{t(|x|)}}[M(x,r) = 1] \geq \frac{2}{3}$ and
   - for any $x \notin L$, $\mathbb{P}_{r \in_R \{0,1\}^{t(|x|)}}[M(x,r) = 0] = 1$

   where $t(n)$ denotes the time complexity of M.

2. Let **co−RP** consist of those languages L for which there is some efficient randomized TM M such that
   - for any $x \in L$, $\mathbb{P}_{r \in_R \{0,1\}^{t(|x|)}}[M(x,r) = 1] = 1$ and
   - for any $x \notin L$, $\mathbb{P}_{r \in_R \{0,1\}^{t(|x|)}}[M(x,r) = 0] \geq \frac{2}{3}$

   where $t(n)$ denotes the time complexity of M.

Let $\mathbf{ZPP} \coloneqq \mathbf{RP} \cap \mathbf{co{-}RP}$.

Note 3.11.6. $L \in \mathbf{ZPP}$ if there exists a randomized algorithm that runs in expected polynomial time and never errs.

3.12 Lecture 20

Theorem 3.12.1. *If there exist $L \in \mathsf{DTIME}(2^{O(n)})$ and $\gamma > 0$ such that L requires circuits of size at least $2^{\gamma n}$, then $\mathbf{P} = \mathbf{BPP}$.*

Note 3.12.2. Recall that any algorithm running in time $t(n)$ can be simulated by circuits of size $t(n)^2$. Let $L \coloneqq \{1^n \mid n \in \mathbb{N}$, the TM represented by the number n in binary halts when its input equals $n\}$. Then L is undecidable but has polynomial size circuits.

Theorem 3.12.3 (Adleman). *Every $L \in \mathbf{BPP}$ has polynomial size circuits. In other words, $\mathbf{BPP} \subset \mathbf{P}/\mathrm{poly}$.*

Proof. Let $L(M) = L$ where M is a randomized TM that runs in polynomial time. Let $n \in \mathbb{N}$. Remark 3.11.4 implies that there is some random bit r_n such that for any string x of size n, $M(x, r_n) = \chi_L(x)$. From this, we can obtain a circuit C_{r_n} of size quadratic in the running time of M such that

$$C_{r_n}(x) = M(x, r_n) = \chi_L(x)$$

for each x of size n. □

Corollary 3.12.4. $\mathbf{BPP} \subset \mathbf{EXP}$.

Definition 3.12.5 (Polynomial hierarchy).

1. For any $i \in \mathbb{N}$, Σ_p^i is the class of languages L for which there exist a polynomial-time computable predicate P and polynomials $p_1(\cdot), \ldots, p_i(\cdot)$ such that

$$x \in L \iff \exists \overline{x_1} \in \{0,1\}^{p_1(|x|)} \forall \overline{x_2} \in \{0,1\}^{p_2(|x|)} \cdots \exists/\forall \overline{x_i} \in \{0,1\}^{p_i(|x|)} (P(x, \overline{x_1}, \ldots, \overline{x_i}) = 1).$$

2. For any $i \in \mathbb{N}$, Π_p^i is the class of languages L for which there exists a polynomial-time computable predicate P and polynomials $p_1(\cdot), \ldots, p_i(\cdot)$ such that

$$x \in L \iff \forall \overline{x_1} \in \{0,1\}^{p_1(|x|)} \exists \overline{x_2} \in \{0,1\}^{p_2(|x|)} \cdots \exists/\forall \overline{x_i} \in \{0,1\}^{p_i(|x|)} (P(x, \overline{x_1}, \ldots, \overline{x_i}) = 1).$$

The *polynomial hierarchy* is the set of languages $\mathbf{PH} \equiv \bigcup_{i \in \mathbb{N}} \Sigma_p^i$.

Aside. For a given (finite) *vocabulary* σ, let $\mathsf{STRUC}[\sigma]$ denote the set of all finite structures of σ equipped with a total ordering. Consider the set

$$\mathcal{B} \coloneqq \{I_b \mid I_b : \mathsf{STRUC}[\sigma] \to \{0,1\},\ \sigma \text{ is a vocabulary}\}.$$

For each function $I_b : \mathsf{STRUC}[\sigma] \to \{0,1\}$, let

$$\mathcal{L}_{I_b} = \{A \in \mathsf{STRUC}[\sigma] \mid I_b(A) = 1\},$$

called a *Boolean query.*

Consider the set $\mathcal{B} \coloneqq \{\mathcal{L}_{I_b} \mid I_b \in \mathcal{B}\}$ of all Boolean queries. Let $\mathbf{SO}$ denote the set of all Boolean queries expressible in second-order logic. That is, $\mathcal{L}_{I_b} \in \mathbf{SO}$ iff there is some second-order sentence φ such that

$$I_b(A) = 1 \iff A \models \varphi.$$

It is known that $\mathbf{SO} = \mathbf{PH}$. This is a well-known result of so-called *descriptive complexity theory*, a branch of finite model theory.

Example 3.12.6. We see that $\mathsf{MAX{-}CLIQUE} \in \Sigma_p^2$ because it is defined by the formula "there exists a choice of vertices V_1 such that for any choice of vertices V_2, V_1 is a clique of size k and V_2 is either not a clique or of size smaller than k."

3.13 Lecture 21

Note 3.13.1.

1. $\Sigma_p^0 = \mathbf{P}$.
2. $\Sigma_p^1 = \mathbf{NP}$.
3. $\Sigma_p^k \subset \Sigma_p^{k+1} \cap \Pi_p^{k+1}$.

4. $\Pi_p^k \subset \Sigma_p^{k+1} \cap \Pi_p^{k+1}$.

5. $\mathsf{co}\text{−}\Sigma_p^k = \Pi_p^k$.

Lemma 3.13.2. *If $k > 0$ and $\Sigma_p^k = \Pi_p^k$, then $\Sigma_p^{k+1} = \Sigma_p^k$.*

Proof. For convenience, let $k = 1$. Note that $L \in \Sigma_p^2$ if and only if some formula

$$\varphi(x) \coloneqq \exists \overline{y_1} \forall \overline{y_2} (P(x, \overline{y_1}, \overline{y_2}))$$

defines L. But, by assumption, $\varphi(x)$ is equivalent to some formula $\exists \overline{y_1} \exists \overline{y_2} (P'(x, \overline{y_1}, \overline{y_2}))$. □

Theorem 3.13.3 (Sipser-Gács). $\mathbf{BPP} \subset \Sigma_p^2 \cap \Pi_p^2$.

Proof. Since $\mathbf{BPP} = \mathsf{co}\text{−}\mathbf{BPP}$, it suffices to show that $\mathbf{BPP} \subset \Sigma_p^2$. If $L \in \mathbf{BPP}$, then there exists an efficient algorithm A such that $\mathbb{P}_{r \in_R \{0,1\}^{\mathrm{poly}(n)}}[A(x,r) = \chi_L(x)] \geq \frac{2}{3}$ where n denotes $|x|$. Define A' to run $A(x, r_1), \ldots, A(x, r_s)$ and take the majority. Then A' uses st random bits, and

$$\mathbb{P}_{r_1,\ldots,r_s}[A'(x, r_1, \ldots, r_s) = \chi_L(x)] \geq 1 - 2^{-s(n)}.$$

By choosing $s \gg 10t^2$, we see that $\mathbb{P}[A'(x, \overline{r}) = \chi_L(x)] \geq 1 - \frac{1}{100m^2}$ where m denotes the number of random bits used.

Claim. $x \in L \iff \exists \overline{y_1}, \ldots, \overline{y_m} \in \{0,1\}^m \forall \overline{z} \in \{0,1\}^m \bigvee_{j=1}^m A'(x, \overline{y_j} \oplus \overline{z}) = 1$.

Proof.
($\Longrightarrow$) Suppose that $x \in L$. It suffices to show that

$$\mathbb{P}_{\overline{y_1},\ldots,\overline{y_m}}\left[\exists \overline{z} \in \{0,1\}^m \bigwedge_{j=1}^m A'(\overline{x}, \overline{y_j} \oplus \overline{z}) \neq 1\right] < 1.$$

Note that $\mathbb{P}_{\overline{y_1},\ldots,\overline{y_m}}\left[\bigwedge_{j=1}^m A'(\overline{x}, \overline{y_j} \oplus \overline{z}) \neq 1\right] \leq \frac{1}{(100m^2)^m}$ for any $\overline{z} \in \{0,1\}^m$. Therefore,

$$\mathbb{P}_{\overline{y_1},\ldots,\overline{y_m}}\left[\exists \overline{z} \in \{0,1\}^m \bigwedge_{j=1}^m A'(\overline{x}, \overline{y_j} \oplus \overline{z}) \neq 1\right] \leq \frac{2^m}{(100m^2)^m} < 1.$$

($\Longleftarrow$) Suppose that $x \notin L$. Fix $y_1, \ldots, y_m$. Note that $\mathbb{P}_{z \in_R \{0,1\}^*}[A(x, y_j \oplus z) = 1] \leq \frac{1}{100m^2}$ for each $j = 1, \ldots, m$. This implies that

$$\mathbb{P}_{z \in_R \{0,1\}^*}[\bigvee_{j=1}^m A(x, y_j \oplus z) = 1] \leq \frac{1}{100m^2} \leq \frac{m}{100m^2} = \frac{1}{100m} < 1.$$

□

□

3.14 Lecture 22

Definition 3.14.1.

1. Let $V, P : \{0,1\}^* \to \{0,1\}^*$ be mappings and x a binary string. Let $r \in_R \{0,1\}^*$. A *k-round (randomized) interaction of V and P on x and r* is the sequence of length k consisting of the following strings.

$$\begin{aligned} a_1 &= V(x, r) \\ a_2 &= P(x, a_1) \\ &\vdots \\ a_{2i+1} &= V(x, r, a_1, \ldots, a_{2i}) \\ a_{2i+2} &= P(x, a_1, \ldots, a_{2i+1}) \end{aligned}$$

Let out $\langle V, P\rangle(x, r)$ denote the final string of this sequence. We call V a *verifier* and P a *prover.*

2. Let $k : \mathbb{N} \to \mathbb{N}$ be polynomial computable. A language L has a *has a k-round (randomized) interactive protocol* or *lies in the class* $\mathsf{IP}[k]$ if there exists a randomized TM V that V is polynomial in its first input and
   (a) (completeness) if $x \in L$, then there exists a prover P such that $\langle V, P\rangle(x)$ is $k(|x|)$-round interaction and
   $$\mathbb{P}_{r \in_R \in \{0,1\}^{\mathrm{poly}(|x|)}}[\text{out } \langle V, P\rangle(x, r) = 1] \geq \frac{2}{3}$$
   and
   (b) (soundness) if $x \notin L$, then for any prover P such that $\langle V, P\rangle(x)$ is $k(|x|)$-round interaction,
   $$\mathbb{P}_{r \in_R \in \{0,1\}^{\mathrm{poly}(|x|)}}[\text{out } \langle V, P\rangle(x, r) = 1] \leq \frac{1}{3}.$$
3. Let $\mathbf{IP} := \bigcup_{k \in \mathbb{N}} \mathsf{IP}[n^k]$.

Example 3.14.2. Let $\mathsf{NIP} := \{\langle G_1, G_2\rangle \mid G_1 \text{ and } G_2 \text{ are non-isomorphic graphs}\}$. The following interaction shows that $\mathsf{NIP} \in \mathbf{IP}$.

V : Pick $i \in \{1, 2\}$ uniformly randomly. Randomly permute the vertices of G_i to get a new isomorphic graph H. Send H to P.
P : If H is not isomorphic to one of G_1 and G_2, then select the other graph.
Otherwise, select one of G_1 and G_2 by flipping a coin. Let G_j denote the selected graph. Send j to V.
V : Pick $i' \in \{1, 2\}$ uniformly randomly. Randomly permute the vertices of $G_{i'}$ to get a new isomorphic graph H'. Send H' to P.
P : If H' is not isomorphic to one of G_1 and G_2, then select the other graph.
Otherwise, select one of G_1 and G_2 by flipping a coin. Let $G_{j'}$ denote the selected graph. Send j' to V.
V : Accept if both $i = j$ and $i' = j'$. Reject otherwise.

3.15 Lecture 23

Proposition 3.15.1. $\mathbf{IP}$ *is closed under complementation.*

Remark 3.15.2. The prover P of a k-round interaction can be assumed, without loss of generality, to decide languages in and only in PSPACE. As a result, $\mathbf{IP} \subset \mathsf{PSPACE}$.

Theorem 3.15.3. $\mathsf{PSPACE} \subset \mathbf{IP}$.

Corollary 3.15.4. $\mathsf{PSPACE} = \mathbf{IP}$.

Definition 3.15.5. Let A and B be sets of size 2^n and 2^k, respectively. A set of functions $\mathcal{H} := \{h_1, \dots, h_t\}$ from A to B is *pairwise independent* if for any distinct $x, x' \in A$ and any $y, y' \in B$,
$$\mathbb{P}_{h \in_R \mathcal{H}}[h(x) = y \wedge h(x') = y'] = \frac{1}{2^{2k}} = \frac{1}{|B|^2}.$$
An element of such a set is called a *(pairwise independent) hash function.*

Example 3.15.6. The set of all functions $A \to B$ is a pairwise independent set of size $|B|^{|A|}$.

Definition 3.15.7. Let $q = 2^n$. For each $(s, t) \in \mathbb{F}_q \times \mathbb{F}_q$, define $h_{s,t} : \mathbb{F}_q \to \mathbb{F}_q$ by $h_{s,t}(a) = a \cdot s + t$. If $x, y, x', y' \in \mathbb{F}_q$ with $x \neq x'$, then the system of equations
$$sx + t = y$$
$$sx' + t = y'$$
is satisfied $\iff \begin{bmatrix} x & 1 \\ x' & 1 \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} y \\ y' \end{bmatrix} \iff \begin{bmatrix} x & 1 \\ x' & 1 \end{bmatrix}^{-1} \begin{bmatrix} y \\ y' \end{bmatrix} = \begin{bmatrix} s \\ t \end{bmatrix}$. Therefore,
$$\mathbb{P}_{(s,t) \in_R \mathbb{F}_q \times \mathbb{F}_q}[h_{s,t}(x) = y \wedge h_{s,t}(x') = y'] = \frac{1}{q^2},$$
which proves that $h_{s,t}$ is a hash function.

3.16 Lecture 24

Proposition 3.16.1. *Let G and H be graphs.*

1. *If $G \ncong H$, then $\operatorname{Aut}(G \cup H) \cong \operatorname{Aut}(G) \times \operatorname{Aut}(H)$.*
2. *There is some $k \in \mathbb{N}$ such that $|\operatorname{Aut}(G \cup H)| \geq 2k$ whenever $G \cong H$ and $|\operatorname{Aut}(G \cup H)| \leq k$ whenever $G \ncong H$.*

Example 3.16.2 (Goldwasser-Sipser set bound protocol). Let $S \subset \{0,1\}^m$ and $K \in \mathbb{N}$. We want to construct a verifier V and prover P such that

- V can efficiently check whether any given x belongs to S, and
- it is guaranteed that either $|S| \leq \frac{K}{2}$ or $|S| \geq K$.

To do this, choose $l \in \mathbb{N}$ such that $2^{l-2} \leq K \leq 2^{l-1}$ and $|S| \leq 2^{l-1}$. Let $\mathcal{H}$ denote a pairwise independent set of mappings $\{0,1\}^m \to \{0,1\}^l$.

Have V randomly choose $h \in \mathcal{H}$ and $y \in \{0,1\}^l$ and then send (h,y) to P. Next, have P send $x \in_R \{0,1\}^m$ to V. Finally, have V accept if and only if $x \in S$ and $h(x) = y$.

Case 1: Suppose that $|S| \leq \frac{K}{2}$.
If $x \in S$, then $\mathbb{P}_{h \in_R \mathcal{H}}[h(x) = y] = \frac{1}{2^l}$. Moreover, if p denotes the quantity $\frac{|K|}{2^l}$, then

$$\mathbb{P}_{h \in_R \mathcal{H}}[\exists x \in S,\ h(x) = y] \leq \frac{|S|}{2^l} \leq \frac{|K|}{2 \cdot 2^l} = \frac{p}{2}.$$

Case 2: Suppose that $|S| \geq K$.
We compute

$$\begin{aligned}
\mathbb{P}_{h \in \mathcal{H}}[\exists x \in S,\ h(x) = y] &\geq \sum_{x \in S} \mathbb{P}[h(x) = y] - \sum_{\substack{x, x' \in S \\ x \neq x'}} \mathbb{P}[h(x) = y \wedge h(x') = y] \\
&\geq \frac{|S|}{2^l} - \underbrace{\frac{|S|(|S|-1)}{2}}_{\binom{|S|}{2}} \cdot \frac{1}{2^{2l}} \\
&= \frac{|S|}{2^l}\left(1 - \frac{|S|-1}{2} \cdot \frac{1}{2^l}\right) \\
&\geq \frac{|S|}{2^l}\left(1 - \frac{|S|}{2 \cdot 2^l}\right) \\
&\geq \frac{K}{2^l}\left(1 - \frac{|S|}{2 \cdot 2^l}\right) \\
&\geq \frac{3}{4} \cdot p.
\end{aligned}$$

3.17 Lecture 25

Lemma 3.17.1 (Sum-check protocol). *Let $n \in \mathbb{N}$ and choose a prime $2^{10n} \leq p \leq 2^{20n}$. Let $\varphi(x_1, \ldots, x_n)$ be a 3cnf formula with m clauses and let $\tilde{\varphi}(x_1, \ldots, x_n)$ be the polynomial obtained from φ by the following translation rules.*

- $\bar{x} \longleftrightarrow (1 - x)$.
- $x \wedge y \longleftrightarrow x \cdot y$.

For any $a_1, \ldots, a_i \in \mathbb{Z}_p$, define

$$S(a_1, \ldots, a_i) = \sum_{x \in \{0,1\}^{n-i}} \tilde{\varphi}(a_1, \ldots, a_i, x) \mod p.$$

For any $K \in \mathbb{N}$, there exists an efficient interactive protocol (V, P) such that

- *if $K = S(a_1, \ldots, a_i)$, then V accepts $\langle \varphi, p \rangle$ with probability 1 and*
- *if $K \neq S(a_1, \ldots, a_i)$, then V rejects $\langle \varphi, p \rangle$ with probability $\geq (1 - \frac{d}{p})^{n-i}$ where $d \coloneqq 3m \leq n^3$.*

3.18 Lecture 26

Proof.
First, we construct (V, P) as follows.

1. $\underline{V}$: If $n = 1$, then compute $\tilde{\varphi}(0) + \tilde{\varphi}(1)$. If this equals K, then *accept*. Otherwise, *reject*.
   If $n > 1$, then let $h_1(x_1) = \sum_{x_2, \ldots, x_n \in \{0,1\}} \tilde{\varphi}(x_1, x_2, \ldots, x_n)$, which is a univariate polynomial of degree at most n^3. Ask P to send $h_1(x_1)$.
2. $\underline{P}$: Return $h_1'(x_1)$ to V where $h_1'(x_1)$ is univariate and has degree at most d.
3. $\underline{V}$: Compute $h_1'(0) + h_1'(1)$. If this equals K, then *reject*.
   Otherwise, choose $a_1 \in_R \mathbb{F}_p$. Recursively apply the same protocol thus far with K replaced with $h_1'(a_1)$ and $\sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \tilde{\varphi}(x_1, x_2, \ldots, x_n)$ replaced with
   $$\sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \tilde{\varphi}(a_1, x_2, \ldots, x_n).$$

Next, we must verify the correctness of (V, P). If $K = \sum_{x_1, \ldots, x_n \in \{0,1\}} \tilde{\varphi}(x_1, \ldots, x_n)$, then have P return $h_i(x_i)$ for each $i = 1, \ldots, n-1$. In this case, V accepts with probability 1.

Now, assume that $K \neq \sum_{x_1, \ldots, x_n \in \{0,1\}} \tilde{\varphi}(x_1, \ldots, x_n)$. If $n = 1$, then clearly V rejects with probability 1. Assume, inductively, that V rejects with high probability for any polynomial of degree $\leq d$ in $n-1$ variables. If $h_1'(x_1) = h_1(x_1)$, then V rejects with probability 1. Assume that $h_1'(x_1) \neq h_1(x_1)$. Note that the polynomial $h_1' - h_1$ is nonzero and has degree at most d. By DeMillo-Lipton, it follows that

$$\mathbb{P}_{a \in_R \mathbb{F}_p}[h_1'(a) \neq h_1(a)] \geq 1 - \frac{d}{p}.$$

Since $s(a) \neq h(a) = \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \tilde{\varphi}(a_1, x_2, \ldots, x_n)$, we see, by our induction hypothesis, that V rejects its recursive input with probability $\geq (1 - \frac{d}{p})^{n-1}$. Thus, V rejects (φ, K, p) with probability

$$\geq \left(1 - \frac{d}{p}\right)^{n-1} \cdot \left(1 - \frac{d}{p}\right) = \left(1 - \frac{d}{p}\right)^n,$$

as desired. ☐

Proposition 3.18.1. *The language of all unsatisfiable 3cnf formulas is* **co−NP***-complete.*

Corollary 3.18.2. **co−NP ⊂ IP***.*

Proof. We can modify our last interactive protocol so that its first round has P send a large enough prime p to V. As a result, we can remove p from the input of (V, P). Notice that φ is unsatisfiable if and only if $\sum_{x \in \{0,1\}^n} \tilde{\varphi}(x_1, \ldots, x_n) = 0$. This, in turn, is true if and only if

$$\sum_{x \in \{0,1\}^n} \tilde{\varphi}(x_1, \ldots, x_n) = 0 \mod p$$

since $0 \leq \sum_{x \in \{0,1\}^n} \tilde{\varphi}(x_1, \ldots, x_n) \leq 2^n$. By our last proposition, we are done. ☐

Theorem 3.18.3. **IP** = PSPACE.

3.19 Final exam review session

Theorem 3.19.1 (Space hierarchy). *If* $f : \mathbb{N} \to \mathbb{N}$ *satisfies* $f(n) \geq \log n$*, then* $\mathsf{DSPACE}(f(n)) \subsetneq \mathsf{DSPACE}(f^2(n))$.

Example 3.19.2. Since $\log^2(n) \leq p(n)$ for some polynomial $p : \mathbb{N} \to \mathbb{N}$, the space hierarchy theorem implies that $\mathbf{L} \subsetneq \mathsf{PSPACE}$.