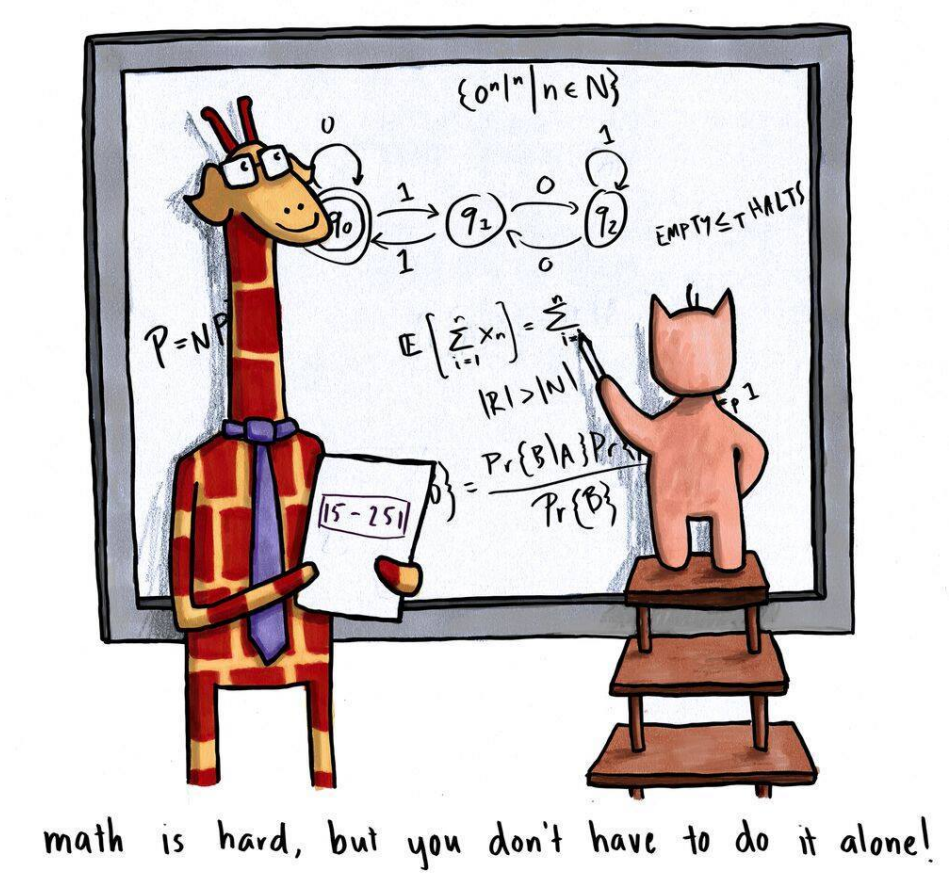
CMU 15-251, Fall 2017
Great Ideas in Theoretical Computer Science

Course Notes: Solutions to Exercises



math is hard, but you don't have to do it alone!

November 17, 2017

Please send comments and corrections to Anil Ada (aada@cs.cmu.edu).

Foreword

These notes are based on the lectures given by Anil Ada and Ariel Procaccia for the Fall 2017 edition of the course 15-251 "Great Ideas in Theoretical Computer Science" at Carnegie Mellon University. They are also closely related to the previous editions of the course, and in particular, lectures prepared by Ryan O'Donnell.

WARNING: The purpose of these notes is to complement the lectures. These notes do *not* contain full explanations of all the material covered during lectures. In particular, the intuition and motivation behind many concepts and proofs are explained during the lectures and not in these notes.

There are various versions of the notes that omit certain parts of the notes. Go to the course webpage to access all the available versions.

In the main version of the notes (i.e. the main document), each chapter has a preamble containing the chapter structure and the learning goals. The preamble may also contain some links to concrete applications of the topics being covered. At the end of each chapter, you will find a short quiz for you to complete before coming to recitation, as well as hints to selected exercise problems.

Note that some of the exercise solutions are given in full detail, whereas for others, we give all the main ideas, but not all the details. We hope the distinction will be clear.

Acknowledgements

The course 15-251 was created by Steven Rudich many years ago, and we thank him for creating this awesome course. Here is the webpage of an early version of the course:
http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15251-s04/Site/.
Since then, the course has evolved. The webpage of the current version is here:
http://www.cs.cmu.edu/~15251/.

Thanks to the previous instructors of 15-251, who have contributed a lot to the development of the course: Victor Adamchik, Luis von Ahn, Anupam Gupta, Venkatesan Guruswami, Bernhard Haeupler, John Lafferty, Ryan O'Donnell, Ariel Procaccia, Daniel Sleator and Klaus Sutner.

Thanks to Eric Bae, Seth Cobb, Teddy Ding, Ellen Kim, Aditya Krishnan, Xinran Liu, Matthew Salim, Ticha Sethapakdi, Vanessa Siriwalothakul, Natasha Vasthare, Jenny Wang, Ling Xu, Ming Yang, Stephanie You, Xingjian Yu and Nancy Zhang for sending valuable comments and corrections on an earlier draft of the notes. And thanks to Darshan Chakrabarti, Emilie Guermeur, Udit Ranasaria, Rosie Sun and Wynne Yao for sending valuable comments and corrections on the current draft.

Contents

Chapter 1

Strings and Encodings

Exercise 1.1 (Structural induction on words)**.**
Let language $L \subseteq \{0, 1\}^*$ be recursively defined as follows:

- $\epsilon \in L$;
- if $x, y \in L$, then $0x1y0 \in L$.

Show, using (structural) induction, that for any word $w \in L$, the number of 0's in w is exactly twice the number of 1's in w.

Solution. Let $\mathbf{0}(w)$ denote the number of 0's in w and let $\mathbf{1}(w)$ denote the number of 1's in w. Given L as defined above, the question asks us to show that for any $w \in L$, $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$. We will do so by structural induction.[1]

The base case corresponds to $w = \epsilon$, and in this case, $\mathbf{0}(w) = \mathbf{1}(w) = 0$, and therefore $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$ holds.

To carry out the induction step, consider an arbitrary word $w \neq \epsilon$ in L. Then by the definition of L, we know that there exists x and y in L such that $w = 0x1y0$. Furthermore, by induction hypothesis,

$$\mathbf{0}(x) = 2 \cdot \mathbf{1}(x) \tag{1.1}$$

and

$$\mathbf{0}(y) = 2 \cdot \mathbf{1}(y). \tag{1.2}$$

We are done once we show $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$. We establish this via the following chain of equalities:

$$\begin{aligned}
\mathbf{0}(w) &= 2 + \mathbf{0}(x) + \mathbf{0}(y) && \text{since } w = 0x1y0 \\
&= 2 + 2 \cdot \mathbf{1}(x) + 2 \cdot \mathbf{1}(y) && \text{by (1.1) and (1.2)} \\
&= 2 \cdot (1 + \mathbf{1}(x) + \mathbf{1}(y)) \\
&= 2 \cdot \mathbf{1}(w).
\end{aligned}$$

■

Exercise 1.2 (Can you distribute star over intersection?)**.**
Prove or disprove: If $L_1, L_2 \subseteq \{\mathtt{a}, \mathtt{b}\}^*$ are languages, then $(L_1 \cap L_2)^* = L_1^* \cap L_2^*$.

Solution. We disprove the statement by providing a counterexample. Let $L_1 = \{\mathtt{a}\}$ and $L_2 = \{\mathtt{aa}\}$. Then $L_1 \cap L_2 = \emptyset$, and so $(L_1 \cap L_2)^* = \{\epsilon\}$. On the other hand, $L_1^* \cap L_2^* = L_2^* = \{\mathtt{aa}\}^*$. ■

Exercise 1.3 (Can you interchange star and reversal?)**.**
Is it true that for any language L, $(L^*)^R = (L^R)^*$? Prove your answer.

Solution. We will prove that for any language L, $(L^*)^R = (L^R)^*$. To do this, we will first argue $(L^*)^R \subseteq (L^R)^*$ and then argue $(L^R)^* \subseteq (L^*)^R$.

To show the first inclusion, it suffices to show that any $w \in (L^*)^R$ is also contained in $(L^R)^*$. We do so now. Take an arbitrary $w \in (L^*)^R$. Then for some $n \in \mathbb{N}$, $w = (u_1 u_2 \dots u_n)^R$, where $u_i \in L$ for each i. Note that $w = (u_1 u_2 \dots u_n)^R = u_n^R u_{n-1}^R \dots u_1^R$, and $u_i^R \in L^R$ for each i. Therefore $w \in (L^R)^*$.

To show the second inclusion, it suffices to show that any $w \in (L^R)^*$ is also contained in $(L^*)^R$. We do so now. Take an arbitrary $w \in (L^R)^*$. This means that for some $n \in \mathbb{N}$, $w = v_1 v_2 \dots v_n$, where $v_i \in L^R$ for each i. For each i, define $u_i = v_i^R$ (and so $u_i^R = v_i$). Note that each $u_i \in L$ because $v_i \in L^R$. We can now rewrite w as $w = u_1^R u_2^R \dots u_n^R$, which is equal to $(u_n u_{n-1} \dots u_1)^R$. Since each $u_i \in L$, this shows that $w \in (L^*)^R$.

Since we have shown both $(L^*)^R \subseteq (L^R)^*$ and $(L^R)^* \subseteq (L^*)^R$, we conclude that $(L^*)^R = (L^R)^*$. ■

[1] This means that implicitly, the parameter being inducted on is the number of applications of the recursive rule to create a word w in L.

Exercise 1.4 (Unary encoding of integers)**.**
Describe an encoding of $\mathbb{Z}$ using the alphabet $\Sigma = \{1\}$.

Solution. Let $\text{Enc} : \mathbb{Z} \to \{1\}^*$ be defined as follows:

$$\text{Enc}(x) = \begin{cases} 1^{2x-1} & \text{if } x > 0, \\ 1^{-2x} & \text{if } x \leq 0. \end{cases}$$

This solution is inspired by thinking of a bijection between integers and naturals. Indeed, the function $f : \mathbb{Z} \to \mathbb{N}$ defined by

$$f(x) = \begin{cases} 2x - 1 & \text{if } x > 0, \\ -2x & \text{if } x \leq 0, \end{cases}$$

is such a bijection. ■

Chapter 2

Deterministic Finite Automata

Exercise 2.1 (Draw DFAs)**.**
For each language below (over the alphabet $\Sigma = \{0, 1\}$), draw a DFA recognizing it.

(a) $\{110, 101\}$

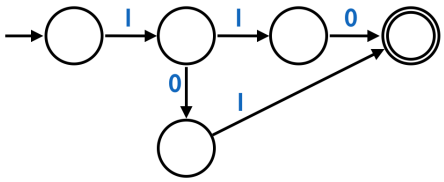
(b) $\{0, 1\}^* \backslash \{110, 101\}$

(c) $\{x \in \{0, 1\}^* : x \text{ starts and ends with the same bit}\}$

(d) $\{\epsilon, 110, 110110, 110110110, \ldots\}$

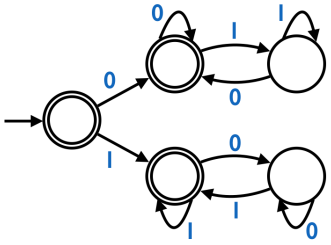
(e) $\{x \in \{0, 1\}^* : x \text{ contains } 110 \text{ as a substring}\}$

Solution. (a) Below, all missing transitions go to a rejecting sink state.

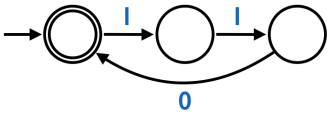


(b) Take the DFA above and flip the accepting and rejecting states.

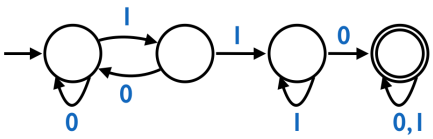
(c)



(d) Below, all missing transitions go to a rejecting sink state.



(e)



■

Exercise 2.2 (Finite languages are regular)**.**
Let L be a finite language, i.e., it contains a finite number of words . Show that there is a DFA recognizing L.

Solution. Sorry, we currently do not have a solution for this exercise. ■

Exercise 2.3 (Equal number of 01's and 10's)**.**
Is the language

$\{w \in \{0,1\}^* : w$ contains an equal number of occurrences of 01 and 10 as substrings.$\}$

regular?

Solution. The answer is yes because the language is exactly same as the language in Exercise (Draw DFAs), part (c). ■

Exercise 2.4 ($a^n b^n c^n$ is not regular)**.**
Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Prove that $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \mathbb{N}\}$ is not regular.

Solution. Our goal is to show that $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that L is regular.

Since L is regular, by definition, there is some deterministic finite automaton M that recognizes L. Let k denote the number of states of M. For $n \in \mathbb{N}$, let r_n denote the state that M reaches after reading $\mathtt{a}^n$ (i.e., $r_n = \delta(q_0, \mathtt{a}^n)$). By the pigeonhole principle, we know that there must be a repeat among $r_0, r_1, \ldots, r_k$. In other words, there are indices $i, j \in \{0, 1, \ldots, k\}$ with $i \neq j$ such that $r_i = r_j$. This means that the string $\mathtt{a}^i$ and the string $\mathtt{a}^j$ end up in the same state in M. Therefore $\mathtt{a}^i w$ and $\mathtt{a}^j w$, *for any* string $w \in \{0,1\}^*$, end up in the same state in M. We'll now reach a contradiction, and conclude the proof, by considering a particular w such that $\mathtt{a}^i w$ and $\mathtt{a}^j w$ end up in different states.

Consider the string $w = \mathtt{b}^i\mathtt{c}^i$. Then since M recognizes L, we know $\mathtt{a}^i w = \mathtt{a}^i\mathtt{b}^i\mathtt{c}^i$ must end up in an accepting state. On the other hand, since $i \neq j$, $\mathtt{a}^j w = \mathtt{a}^j\mathtt{b}^i\mathtt{c}^i$ is not in the language, and therefore cannot end up in an accepting state. This is the desired contradiction. ■

Exercise 2.5 ($c^{251}a^n b^{2n}$ is not regular)**.**
Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Prove that $L = \{\mathtt{c}^{251}\mathtt{a}^n\mathtt{b}^{2n} : n \in \mathbb{N}\}$ is not regular.

Solution. Our goal is to show that $L = \{\mathtt{c}^{251}\mathtt{a}^n\mathtt{b}^{2n} : n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that L is regular.

Since L is regular, by definition, there is some deterministic finite automaton M that recognizes L. Let k denote the number of states of M. For $n \in \mathbb{N}$, let r_n denote the state that M reaches after reading $\mathtt{c}^{251}\mathtt{a}^n$. By the pigeonhole principle, we know that there must be a repeat among $r_0, r_1, \ldots, r_k$. In other words, there are indices $i, j \in \{0, 1, \ldots, k\}$ with $i \neq j$ such that $r_i = r_j$. This means that the string $\mathtt{c}^{251}\mathtt{a}^i$ and the string $\mathtt{c}^{251}\mathtt{a}^j$ end up in the same state in M. Therefore $\mathtt{c}^{251}\mathtt{a}^i w$ and $\mathtt{c}^{251}\mathtt{a}^j w$, *for any* string $w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$, end up in the same state in M. We'll now reach a contradiction, and conclude the proof, by considering a particular w such that $\mathtt{c}^{251}\mathtt{a}^i w$ and $\mathtt{c}^{251}\mathtt{a}^j w$ end up in different states.

Consider the string $w = \mathtt{b}^{2i}$. Then since M recognizes L, we know $\mathtt{c}^{251}\mathtt{a}^i w = \mathtt{c}^{251}\mathtt{a}^i\mathtt{b}^{2i}$ must end up in an accepting state. On the other hand, since $i \neq j$, $\mathtt{c}^{251}\mathtt{a}^j w = \mathtt{c}^{251}\mathtt{a}^j\mathtt{b}^{2i}$ is not in the language, and therefore cannot end up in an accepting state. This is the desired contradiction. ■

Exercise 2.6 (Are regular languages closed under complementation?)**.**
Is it true that if L is regular, than its complement $\Sigma^* \backslash L$ is also regular? In other words, are regular languages *closed* under the complementation operation?

Solution. Yes. If L is regular, then there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L. The complement of L is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, Q \backslash F)$. Take a moment to observe that this exercise allows us to say that a language is regular *if and only if* its complement is regular. Equivalently, a language is not regular if and only if its complement is not regular. ■

Exercise 2.7 (Are regular languages closed under subsets?)**.**
Is it true that if $L \subseteq \Sigma^*$ is a regular language, then any $L' \subseteq L$ is also a regular language?

Solution. No. For example, $L = \Sigma^*$ is a regular language (construct a single state DFA in which the state is accepting). On the other hand, by Theorem ($\mathtt{0}^n\mathtt{1}^n$ is not regular), $\{\mathtt{0}^n\mathtt{1}^n : n \in \mathbb{N}\} \subseteq \Sigma^*$ is not regular. ■

Exercise 2.8 (Direct proof that regular languages are closed under difference)**.**
Give a direct proof (without using the fact that regular languages are closed under complementation, union and intersection) that if L_1 and L_2 are regular languages, then $L_1 \backslash L_2$ is also regular.

Solution. The proof is very similar to the proof of Theorem (Regular languages are closed under union). The only difference is the definition of F'', which now needs to be defined as

$$F'' = \{(q, q') : q \in F \text{ and } q' \in Q' \backslash F'\}.$$

The argument that $L(M'') = L(M) \backslash L(M')$ needs to be slightly adjusted in order to agree with F''. ■

Exercise 2.9 (Finite vs infinite union)**.**

(a) Suppose $L_1, \ldots, L_k$ are all regular languages. Is it true that their union $\bigcup_{i=0}^{k} L_i$ must be a regular language?

(b) Suppose $L_0, L_1, L_2, \ldots$ is an infinite sequence of regular languages. Is it true that their union $\bigcup_{i \geq 0} L_i$ must be a regular language?

Solution. In part (a), we are asking whether a finite union of regular languages is regular. The answer is yes, and this can be proved using induction, with the base case corresponding to Theorem (Regular languages are closed under union). In part (b), we are asking whether a countably infinite union of regular languages is regular. The answer is no. First note that any language of cardinality 1 is regular, i.e., $\{w\}$ for any $w \in \Sigma^*$ is a regular language. In particular, for any $n \in \mathbb{N}$, the language $L_n = \{\mathtt{0}^n\mathtt{1}^n\}$ of cardinality 1 is regular. But

$$\bigcup_{n \geq 0} L_n = \{\mathtt{0}^n\mathtt{1}^n : n \in \mathbb{N}\}$$

is not regular. ■

Exercise 2.10 (Union of irregular languages)**.**
Suppose L_1 and L_2 are not regular languages. Is it always true that $L_1 \cup L_2$ is not a regular language?

Solution. The answer is no. Consider $L = \{\mathtt{0}^n\mathtt{1}^n : n \in \mathbb{N}\}$, which is a non-regular language. Furthermore, the complement of L, which is $\overline{L} = \Sigma^* \backslash L$, is non-regular. This is because regular languages are closed under complementation (Exercise (Are regular languages closed under complementation?)), so if $\overline{L}$ was regular, then $\overline{\overline{L}} = L$ would also have to be regular. The union of L and $\overline{L}$ is Σ^*, which is a regular language. ■

Exercise 2.11 (Regularity of suffixes and prefixes)**.**
Suppose $L \subseteq \Sigma^*$ is a regular language. Show that the following languages are also regular:

$$\text{SUFFIXES}(L) = \{x \in \Sigma^* : yx \in L \text{ for some } y \in \Sigma^*\},$$
$$\text{PREFIXES}(L) = \{y \in \Sigma^* : yx \in L \text{ for some } x \in \Sigma^*\}.$$

Solution. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing L. Define the set

$$S = \{s \in Q : \exists y \in \Sigma^* \text{ such that } \delta(q_0, y) = s\}.$$

Now we define a DFA for each $s \in S$ as follows: $M_s = (Q, \Sigma, \delta, s, F)$. Observe that

$$\text{SUFFIXES}(L) = \bigcup_{s \in S} L(M_s).$$

Since $L(M_s)$ is regular for all $s \in S$ and S is a finite set, using Exercise (Finite vs infinite union) part (a), we can conclude that SUFFIXES(L) is regular.

For the second part, define the set

$$R = \{r \in Q : \exists x \in \Sigma^* \text{ such that } \delta(r, x) \in F\}.$$

Now we can define the DFA $M_R = (Q, \Sigma, \delta, q_0, R)$. Observe that this DFA recognizes PREFIXES(L), which shows that PREFIXES(L) is regular. ■

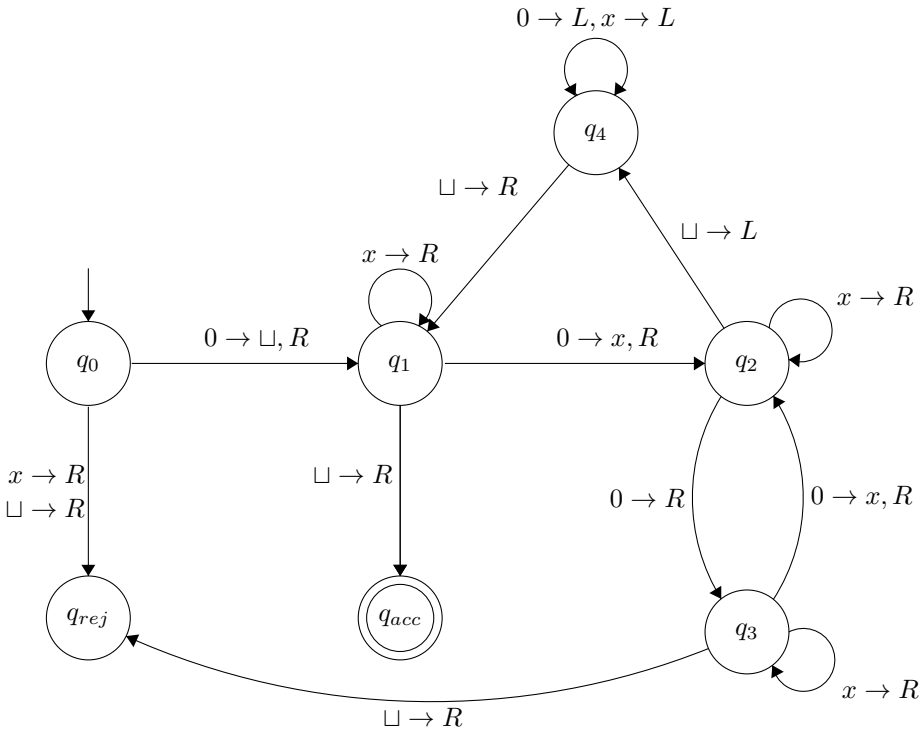
Chapter 3

Turing Machines

Exercise 3.1 (Practice with configurations)**.**

(a) Suppose $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ is a Turing machine. We want you to formally define $\alpha \vdash_M \beta$. More precisely, suppose $\alpha = uqv$, where $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$. Precisely describe β.

(b) Let M denote the Turing machine shown below, which has input alphabet $\Sigma = \{0\}$ and tape alphabet $\Gamma = \{0, x, \sqcup\}$. (Note on notation: A transition label usually has two symbols, one corresponding to the symbol being read, and the other corresponding to the symbol being written. If a transition label has one symbol, the interpretation is that the symbol being read and written is exactly the same.)



We want you to prove that M accepts the input 0000 using the definition on the previous page. More precisely, we want you to write out the computation trace

$$\alpha_0 \vdash_M \alpha_1 \vdash_M \cdots \vdash_M \alpha_T$$

for $M(0000)$. You do not have to justify it; just make sure to get T and $\alpha_0, \ldots, \alpha_T$ correct!

Solution. Part (a): Let $\alpha = uqv$, where $u = u_1 \cdots u_m$ and $v = v_1 \cdots v_n$ (u and v possibly empty). Let v_1' be v_1 if it exists or $\sqcup$ otherwise. Let $\delta(q, v_1') = (q', x, D)$ (where D is either L or R). We write $\alpha \vdash_M \beta$, where β is defined as follows:

- if $D = L, m > 0$, then $\beta = u_1 \ldots u_{m-1} q' u_m x v_2 \ldots v_n$;
- if $D = L, m = 0$, then $\beta = q' \sqcup x v_2 \ldots v_n$;
- if $D = R$, then $\beta = u_1 \ldots u_m x q' v_2 \ldots v_n$.

Part (b): Below is the trace for the execution of the Turing Machine. Read down

first and then to the right.

$q_0 0000$	$q_4 x0x$	xq_4xx
$q_1 000$	$q_4 \sqcup x0x$	q_4xxx
$xq_2 00$	$q_1 x0x$	$q_4 \sqcup xxx$
$x0q_3 0$	$xq_1 0x$	q_1xxx
$x0xq_2$	xxq_2x	xq_1xx
$x0q_4x$	$xxxq_2$	xxq_1x
$xq_4 0x$	xxq_4x	$xxxq_1$
		$xxx \sqcup q_{acc}$

■

Exercise 3.2 (A simple decidable language)**.**
Give a description of the language decided by the TM shown in the example corresponding to Definition (Turing machine).

Solution. The language decided by the TM is

$$L = \{w \in \{a, b\}^* : |w| \geq 2 \text{ and } w_1 = w_2\}.$$

■

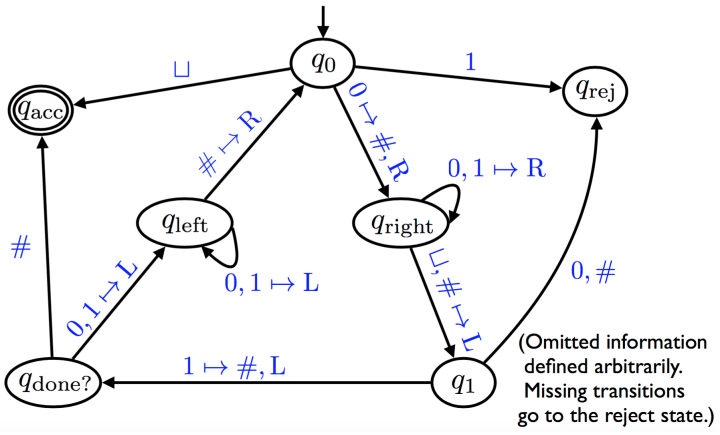
Exercise 3.3 (Drawing TM state diagrams)**.**
For each language below, draw the state diagram of a TM that decides the language. You can use any finite tape alphabet Γ containing the elements of Σ and the symbol $\sqcup$.

(a) $L = \{0^n 1^n : n \in \mathbb{N}\}$, where $\Sigma = \{0, 1\}$.

(b) $L = \{0^n : n \text{ is a nonnegative integer power of } 2\}$, where $\Sigma = \{0\}$.

Solution. Part (a):

$$\Sigma = \{0, 1\} \qquad \Gamma = \{0, 1, \#, \sqcup\}$$



Part (b): See the figure in Exercise (Practice with configurations), part (b). ■

Exercise 3.4 (Decidability is closed under intersection and union)**.**
Let L and K be decidable languages. Show that $L \cap K$ and $L \cup K$ are also decidable by presenting high-level descriptions of TMs deciding them.

Solution. Since L_1 and L_2 are decidable, there are decider TMs M_1 and M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$. To show $L_1 \cup L_2$ is decidable, we present a high-level description of a TM M deciding it:

x: string
$M(x)$:
1. Run $M_1(x)$, if it accepts, accept.
2. Run $M_2(x)$, if it accepts, accept.
3. Reject.

It is pretty clear that this decider works correctly. However, in case you are wondering how in general (with more complicated examples) we would prove that a decider works as desired, here is an example argument.

We want to show that $x \in L_1 \cup L_2$ if and only if it is accepted by the above TM M. If $x \in L_1 \cup L_2$, then it is either in L_1 or in L_2. If it is in L_1, then $M_1(x)$ accepts (since M_1 correctly decides L_1) and therefore M accepts on line 1. If, on the other hand, $x \in L_2$, then $M_2(x)$ accepts. This means that if M does not accept on line 1, then it has to accept on line 2. Either way x is accepted by M. For the converse, assume x is accepted by M. Then it must be accepted on line 1 or line 2. If it is accepted on line 1, then this implies that $M_1(x)$ accepted, i.e., $x \in L_1$. If it is accepted on line 2, then $M_2(x)$ accepted, i.e., $x \in L_2$. So $x \in L_1 \cup L_2$, as desired.

To show $L_1 \cap L_2$ is decidable, we present a high-level description of a TM M deciding it:

x: string
$M(x)$:
1. Run $M_1(x)$ and run $M_2(x)$.
2. If they both accept, accept.
3. Else, reject.

Once again, it is clear that this decider works correctly. ■

Exercise 3.5 (Decidable language based on pi)**.**
Let $L \subseteq \{3\}^*$ be defined as follows: $x \in L$ if and only if x appears somewhere in the decimal expansion of π. For example, the strings ϵ, 3, and 33 are all definitely in L, because

$$\pi = 3.14159265358979323846264 33\ldots$$

Prove that L is decidable. No knowledge in number theory is required to solve this question.

Solution. The important observation is the following. If, for some $m \in \mathbb{N}$, 3^m is not in L, then neither is 3^k for any $k > m$. Additionally, if $3^m \in L$, then so is 3^ℓ for every $\ell < m$. For each $n \in \mathbb{N}$, define

$$L_n = \{3^m : m \leq n\}.$$

Then either $L = L_n$ for some n, or $L = \{3\}^*$.

If $L = L_n$ for some n, then the following TM decides it.

x: string
$M(x)$:
1. If $|x| \leq n$, accept.
2. Else, reject.

If $L = \{3\}^*$, then it is decided by:

x: string
$M(x)$:
1 Accept.

So in all cases, L is decidable. ■

Exercise 3.6 (Practice with decidability through reductions)**.**

(a) Let $L = \{\langle D_1, D_2\rangle : D_1 \text{ and } D_2 \text{ are DFAs with } L(D_1) \subsetneq L(D_2)\}$.[1] Show that L is decidable.

(b) Let $K = \{\langle D\rangle : D \text{ is a DFA that accepts } w^R \text{ whenever it accepts } w\}$, where w^R denotes the *reversal* of w. Show that K is decidable. For this question, you can use the fact given a DFA D, there is an algorithm to construct a DFA D' such that $L(D') = L(D)^R = \{w^R : w \in L(D)\}$.

Solution. Part (a): To show L is decidable, we are going to use the fact that $\text{EMPTY}_{\text{DFA}}$ is decidable (Theorem ($\text{EMPTY}_{\text{DFA}}$ is decidable)) and EQ_{DFA} is decidable (Theorem (EQ_{DFA} is decidable)). Let M_{EMPTY} denote a decider TM for $\text{EMPTY}_{\text{DFA}}$ and let M_{EQ} denote a decider TM for EQ_{DFA}.

A decider for L takes as input $\langle D_1, D_2\rangle$, where D_1 and D_2 are DFAs. It needs to determine if $L(D_1) \subsetneq L(D_2)$ (i.e. accept if $L(D_1) \subsetneq L(D_2)$ and reject otherwise). To determine this we do two checks:

(i) Check whether $L(D_1) = L(D_2)$.

(ii) Check whether $L(D_1) \subseteq L(D_2)$. Observe that this can be done by checking whether $L(D_1) \cap \overline{L(D_2)} = \emptyset$.

Note that $L(D_1) \subseteq L(D_2)$ if and only if $L(D_1) \neq L(D_2)$ and $L(D_1) \cap \overline{L(D_2)} = \emptyset$. Using the closure properties of regular languages, we can construct a DFA D such that $L(D) = L(D_1) \cap \overline{L(D_2)}$. Now the decider for L can be described as follows:

D_1: DFA. D_2: DFA.
$M(\langle D_1, D_2\rangle)$:
1 Construct DFA D as described above.
2 Run $M_{\text{EQ}}(\langle D_1, D_2\rangle)$.
3 If it accepts, reject.
4 Else:
5 Run $M_{\text{EMPTY}}(\langle D\rangle)$
6 If it accepts, accept.
7 Else, reject.

Observe that this machine accepts $\langle D_1, D_2\rangle$ if and only if $M_{\text{EQ}}(\langle D_1, D_2\rangle)$ rejects and $M_{\text{EMPTY}}(\langle D\rangle)$ accepts. In other words, it accepts $\langle D_1, D_2\rangle$ if and only if $L(D_1) \neq L(D_2)$ and $L(D_1) \cap \overline{L(D_2)} = \emptyset$, which is the desired behavior for the machine.

Part (b): We sketch the proof. To show L is decidable, we are going to use the fact that EQ_{DFA} is decidable (Theorem (EQ_{DFA} is decidable)). Let M_{EQ} denote a decider TM for EQ_{DFA}. Observe that $\langle D\rangle$ is in K if and only if $L(D) = L(D)^R$ (prove this part). Using the fact given to us in the problem description, we know that there is a way to construct $\langle D'\rangle$ such that $L(D') = L(D)^R$. Then all we need to do is run $M_{\text{EQ}}(\langle D, D'\rangle)$ to determine whether $\langle D\rangle \in K$ or not. ■

[1] Note on notation: for sets A and B, we write $A \subsetneq B$ if $A \subseteq B$ and $A \neq B$.

Chapter 4

Countable and Uncountable Sets

Exercise 4.1 (Exercise with injections and surjections)**.**
Prove parts (a) and (b) of the above theorem.

Solution. Unfortunately we currently do not have the solution to this exercise. ■

Exercise 4.2 (Proof of the characterization of countably infinite sets)**.**
Prove the above theorem.

Solution. We need to show that A is countably infinite if and only if $|A| = |\mathbb{N}|$, so we will argue the two direction separately.

If $|A| = |\mathbb{N}|$, then $|A| \leq |\mathbb{N}|$, and A is infinite because a finite set cannot be in one-to-one correspondence with an infinite set. Therefore A is countably infinite.

For the other direction, assume A is such that $|A| \leq |\mathbb{N}|$ and A is infinite. Since $|A| \leq |\mathbb{N}|$, there is an injection $f : A \to \mathbb{N}$. This f allows us to define an ordering on A. For $a, b \in A$, write $a < b$ if $f(a) < f(b)$. Using this ordering, we can define a bijection $g : \mathbb{N} \to A$, where $g(n)$ is defined to be the n'th smallest element in A (and we start counting from 0). Since A is infinite, $g(n)$ is well-defined for all n. Clearly g is injective since we cannot have $g(n) = g(n')$ for $n \neq n'$. Furthermore g is surjective because for every $a \in A$, the pre-image is $g^{-1}(a) = |\{x \in A : f(x) < f(a)\}|$. ■

Exercise 4.3 (Practice with countability proofs)**.**
Show that the following sets are countable.

(a) $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$.

(b) The set of all functions $f : A \to \mathbb{N}$, where A is a finite set.

Solution. Part (a): We want to show that $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ is countable. We use the CS method with $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, \#\}$. Note that any element of $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ can be written uniquely as a finite word over Σ (we use the hashtag as a separator between the integers). As an illustration, $(9234851, \text{-}1234, 0) \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ can be encoded as the string `9234851#-1234#0`. Every integer has finite length, so the string encoding is always of finite length.

Part (b): Let S be the set of all functions $f : A \to \mathbb{N}$, where A is a finite set. We want to show that S is countable.

We first make an observation about the elements of S. Take a function $f : A \to \mathbb{N}$, where A is a finite set. Let k be the size of A and let $a_1, a_2, \ldots, a_k$ be its elements. Then f can be uniquely represented by the tuple

$$(f(a_1), f(a_2), \ldots, f(a_k)),$$

where each element of the tuple is an element from $\mathbb{N}$.

We now show that S is countable using the CS method with the alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \#\}$. The observation above shows that any element of S can be uniquely represented with a finite length string where commas are replaced with `#`. (Note that there is no need to put the opening and closing parentheses.) This suffices to conclude that S is countable. ■

Exercise 4.4 (Uncountable sets are closed under supersets)**.**
Prove that if A is uncountable and $A \subseteq B$, then B is also uncountable.

Solution. We want to show that if B is a superset of an uncountable set A, then B must be uncountable.

If A is uncountable, by definition, $|A| > |\mathbb{N}|$. If $A \subseteq B$, then there is a clear injection from A to B (map $a \in A$ to $a \in B$), so $|A| \leq |B|$. Combining this with $|A| > |\mathbb{N}|$, we have $|B| \geq |A| > |\mathbb{N}|$, and therefore B is uncountable. ■

Exercise 4.5 (Practice with uncountability proofs)**.**
Show that the following sets are uncountable.

(a) The set of all bijective functions from $\mathbb{N}$ to $\mathbb{N}$.

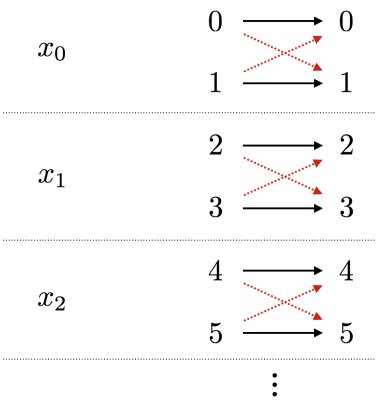
(b) $\{x_1x_2x_3\ldots \in \{1,2\}^\infty : \text{ for all } n \geq 1,\ \sum_{i=1}^{n} x_i \not\equiv 0 \mod 4\}$

Solution. Part (a): Let A be the set of all bijective functions from $\mathbb{N}$ to $\mathbb{N}$. We want to show that A is uncountable, and we will do so by showing that $\{0,1\}^\infty \hookrightarrow A$, establishing $|\{0,1\}^\infty| \leq |A|$.

We now describe this injective mapping. Given $x \in \{0,1\}^\infty$, we map it to a bijection $f_x : \mathbb{N} \to \mathbb{N}$ as follows. Let x_n be the n'th bit of x, and assume the indexing starts from 0. Then for all $n \in \mathbb{N}$,

$$\begin{aligned} &\text{if } x_n = 0, \quad f_x \text{ maps } 2n \text{ to } 2n \text{ and } 2n+1 \text{ to } 2n+1; \\ &\text{if } x_n = 1, \quad f_x \text{ maps } 2n \text{ to } 2n+1 \text{ and } 2n+1 \text{ to } 2n. \end{aligned}$$

The below picture illustrates the construction of f_x. If $x_n = 0$, we pick the black arrows to map $2n$ and $2n+1$, and if $x_n = 1$ we pick the red/dashed arrows to map $2n$ and $2n+1$.



Observe that for any $x \in \{0,1\}^\infty$, the corresponding function f_x is indeed a bijection. It is also clear that if $x \neq x'$, then $f_x \neq f_{x'}$. So this mapping from $\{0,1\}^\infty$ to A is indeed an injection. This completes the proof.

Part (b): Let $A = \{x_1x_2x_3\ldots \in \{1,2\}^\infty : \text{ for all } n \geq 1,\ \sum_{i=1}^{n} x_i \not\equiv 0 \mod 4\}$. We want to show that A is uncountable, and we will do so by identifying a subset of A that is in one-to-one correspondence with $\{0,1\}^\infty$.

Let $a = 22$ and $b = 112$. Define the set $A' = \{1w : w \in \{a,b\}^\infty\}$. Observe that $A' \subseteq A$ (this needs a short argument that we skip). Furthermore, it is clear that there is a bijection between A' and $\{0,1\}^\infty$. So we have identified a subset of A that is in one-to-one correspondence with $\{0,1\}^\infty$, which allows us to conclude that A is uncountable. ■

Chapter 5

Undecidable Languages

Exercise 5.1 (Practice with undecidability proofs)**.**
Show that the following languages are undecidable.

(a) EMPTY-HALTS $= \{\langle M \rangle : M$ is a TM and $M(\epsilon)$ halts$\}$.

(b) FINITE $= \{\langle M \rangle : M$ is a TM that accepts finitely many strings.$\}$

Solution. Part (a): We want to show EMPTY-HALTS is undecidable. To proof is by contradiction, so assume that EMPTY-HALTS is decidable, and let $M_{\text{EMPTY-HALTS}}$ be a decider for it. We will use $M_{\text{EMPTY-HALTS}}$ to show that HALTS is decidable and reach a contradiction. The description of M_{HALTS}, the decider for HALTS, is as follows.

> M: TM. x: string.
> $M_{\text{HALTS}}(\langle M, x \rangle)$:
> 1 Construct the following string, which we call $\langle M' \rangle$.
> 2 "$M'(y)$:
> 3 Run $M(x)$.
> 4 Accept."
> 5 Run $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$.
> 6 If it accepts, accept.
> 7 If it rejects, reject.

To see that this is a correct decider for HALTS, first consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \in$ HALTS, i.e., $M(x)$ halts. By the construction of M', this implies that $M'(y)$ halts (and accepts) for any string y. So $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$ accepts, and our decider above accepts as well. So in this case, the decider gives the correct answer.

Now consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \notin$ HALTS, i.e., $M(x)$ loops. Then for any input y, $M'(y)$ would get stuck on line 3, and would never halt. This means $M_{\text{EMPTY-HALTS}}(\langle M' \rangle)$ rejects, and our decider rejects as well, as desired.

For any input, our decider gives the correct answer, and the proof is complete.

Part(b): Our goal is to show that FINITE is undecidable. To proof is by contradiction, so assume that FINITE is decidable, and let M_{FINITE} be a decider for it. We will use M_{FINITE} to show that HALTS is decidable and reach a contradiction. The description of M_{HALTS}, the decider for HALTS, is as follows.

> M: TM. x: string.
> $M_{\text{HALTS}}(\langle M, x \rangle)$:
> 1 Construct the following string, which we call $\langle M' \rangle$.
> 2 "$M'(y)$:
> 3 Run $M(x)$.
> 4 Accept."
> 5 Run $M_{\text{FINITE}}(\langle M' \rangle)$.
> 6 If it accepts, reject.
> 7 If it rejects, accept.

To see that this is a correct decider for HALTS, first consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \in$ HALTS, i.e., $M(x)$ halts. By the construction of M', this implies that $M'(y)$ accepts for any string y. So $L(M') = \Sigma^*$ (an infinite set), and therefore $M_{\text{FINITE}}(\langle M' \rangle)$ rejects. In this case, our decider for HALTS accepts and gives the correct answer.

Now consider any input $\langle M, x \rangle$ such that $\langle M, x \rangle \notin$ HALTS, i.e., $M(x)$ loops. Then for any input y, $M'(y)$ would get stuck on line 3, and would never halt.

So $L(M') = \emptyset$ (a finite set), and therefore $M_{\text{FINITE}}(\langle M' \rangle)$ accepts. In this case, our decider for HALTS rejects and gives the correct answer.

For any input, our decider gives the correct answer, and the proof is complete. ■

Exercise 5.2 (Practice with reduction definition)**.**
Let $A, B \subseteq \{0, 1\}^*$ be languages. Prove or disprove the following claims.

(a) If $A \leq B$ then $B \leq A$.

(b) If $A \leq B$ and B is regular, then A is regular.

Solution. Part (a): The claim is false. Let A be any decidable language. For example, we can take $A = \emptyset$. The decider for A is a machine that rejects no matter what the input is. Let $B =$ HALTS. Then to establish $A \leq B$, we need to argue that given a decider for HALTS, we can decide $\emptyset$. Since $\emptyset$ is decidable, this is true (and we don't even need to make use of a decider for HALTS). On the other hand, it is **not** true that HALTS $\leq \emptyset$. For the sake of contradiction, if it was true, then this would mean that using a decider for $\emptyset$, we can decide HALTS. And this would imply that HALTS is decidable, a contradiction.

Part (b): The claim is false. Consider $A = \{0^n 1^n : n \in \mathbb{N}\}$ and $B = \emptyset$. We have $A \leq B$ because A is a decidable language (we don't even need to make use of the decider for B). Furthermore, B is regular, but A is not. ■

Exercise 5.3 (Practice with reduction proofs)**.**
Show the following.

(a) ACCEPTS $\leq$ HALTS.

(b) HALTS $\leq$ EQ.

Solution. Part (a): We want to show that ACCEPTS reduces to HALTS. To do this, we assume that HALTS is decidable. Let M_{HALTS} be a decider for HALTS. We now need to construct a TM that decides ACCEPTS (which will make use of M_{HALTS}). Here is the description of the decider:

```
M: TM. x: string.
M_ACCEPTS(⟨M, x⟩):
1 Run M_HALTS(⟨M, x⟩).
2 If it rejects, reject.
3 Else:
4    Run M(x)
5    If it accepts, accept.
6    If it rejects, reject.
```

We now argue that this machine indeed decides ACCEPTS. Note that given $\langle M, x \rangle$, there are three possibilities: $M(x)$ accepts; $M(x)$ rejects, $M(x)$ loops. And $\langle M, x \rangle \in$ ACCEPTS if and only if $M(x)$ accepts.

Let's first consider an input $\langle M, x \rangle$ such that $\langle M, x \rangle \in$ ACCEPTS. Then $M_{\text{HALTS}}(\langle M, x \rangle)$ must accept, i.e. $M(x)$ halts. So the decider above safely simulates $M(x)$ on line 4 and accepts (on line 5).

If the the input is such that $\langle M, x \rangle \notin$ ACCEPTS, then there are two cases. Either $M(x)$ halts but rejects, or $M(x)$ loops. If it is the latter, then $M_{\text{HALTS}}(\langle M, x \rangle)$ rejects, and therefore our decider rejects as well. If on the other hand $M(x)$ halts but rejects, then our decider safely simulates $M(x)$ on line 4 and rejects (on line 6).

Whatever the input is, our decider gives the correct answer. The proof is complete.

Part(b): (This can be considered as an alternative proof of Theorem (EQ is undecidable).) We want to show that HALTS reduces to EQ. To do this, we assume that EQ is decidable. Let M_{EQ} be a decider for EQ. We now need to construct a TM that decides HALTS (which will make use of M_{EQ}). Here is the description of the decider:

M: TM. x: string.
$M_{\text{HALTS}}(\langle M, x\rangle)$:
1 Construct the string $\langle M'\rangle$ where M' is a TM that rejects every input.
2 Construct the following string, which we call $\langle M''\rangle$.
3 "$M''(y)$:
4 Run $M(x)$.
5 Ignore the output and accept."
6 Run $M_{\text{EQ}}(\langle M', M''\rangle)$.
7 If it accepts, reject.
8 If it rejects, accept.

We now argue that this machine indeed decides HALTS. Notice that no matter what the input is, $L(M') = \emptyset$. Let's first consider an input $\langle M, x\rangle$ such that $\langle M, x\rangle \in$ HALTS. Then M'' accepts every input, so $L(M') = \Sigma^*$. In this case, $M_{\text{EQ}}(\langle M', M''\rangle)$ rejects, and so our machine accepts as desired. Next, consider an input $\langle M, x\rangle$ such that $\langle M, x\rangle \notin$ HALTS. Then whatever input is given to M'', it gets stuck in an infinite loop when it runs $M(x)$. So $L(M'') = \emptyset$. In this case $M_{\text{EQ}}(\langle M', M''\rangle)$ accepts, and so our machine rejects, as it should. Thus we have a correct decider for HALTS. ■

Chapter 6

Time Complexity

Exercise 6.1 (Practice with big-O)**.**
Show that $3n^2 + 10n + 30$ is $O(n^2)$.

Solution. Proof 1: To show that $3n^2 + 10n + 30$ is $O(n^2)$, we need to show that there exists $C > 0$ and $n_0 > 0$ such that

$$3n^2 + 10n + 30 \leq Cn^2$$

for all $n \geq n_0$. Pick $C = 4$ and $n_0 = 13$. Note that for $n \geq 13$, we have

$$10n + 30 \leq 10n + 3n = 13n \leq n^2.$$

This implies that for $n \geq 13$ and $C = 4$,

$$3n^2 + 10n + 30 \leq 3n^2 + n^2 = Cn^2.$$

Proof 2: Pick $C = 43$ and $n_0 = 1$. Then for $n \geq 1 = n_0$, we have

$$3n^2 + 10n + 30 \leq 3n^2 + 10n^2 + 30n^2 = 43n^2 = Cn^2.$$

■

Exercise 6.2 (Practice with big-Omega)**.**
Show that $n!^2$ is $\Omega(n^n)$.

Solution. To show $(n!)^2 = \Omega(n^n)$, we'll show that choosing $c = 1$ and $n_0 = 0$ satisfies the definition of Big-Omega. To see this, note that for $n \geq 1$, we have:

$$\begin{aligned}
(n!)^2 &= ((n)(n-1)\cdots(1))\,((n)(n-1)\cdots(1)) && \text{(by definition)}\\
&= (n)(1)(n-1)(2)\cdots(1)(n) && \text{(re-ordering terms)}\\
&= ((n)(1))((n-1)(2))\cdots((1)(n)) && \text{(pairing up consecutive terms)}\\
&\geq (n)(n)\cdots(n) && \text{(by the Claim below)}\\
&= n^n.
\end{aligned}$$

Claim: For $n \geq 1$ and for $i \in \{0, 1, \ldots, n-1\}$,

$$(n - i)(i + 1) \geq n.$$

Proof: The proof follows from the following chain of implications.

$$\begin{aligned}
n - (n-1) - 1 = 0 \quad &\implies \quad n - i - 1 \geq 0 && \text{(since } i \leq n-1\text{)}\\
&\implies \quad i(n - i - 1) \geq 0 && \text{(since } i \geq 0\text{)}\\
&\implies \quad ni - i^2 - i \geq 0\\
&\implies \quad ni - i^2 - i + n \geq n\\
&\implies \quad (n - i)(i + 1) \geq n.
\end{aligned}$$

This completes the proof. ■

Exercise 6.3 (Practice with Theta)**.**
Show that $\log_2(n!) = \Theta(n \log n)$.

Solution. We first show $\log_2 n! = O(n \log n)$. Observe that

$$n! \quad = \quad n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 2 \cdot 1 \quad \leq \quad \underbrace{n \cdot n \cdot n \cdot \cdots \cdot n}_{n \text{ times}} \quad = \quad n^n,$$

as each term on the RHS (i.e. n) is greater than or equal to each term on the LHS. Taking the $\log$ of both sides gives us $\log_2 n! \leq \log_2 n^n = n \log_2 n$ (here, we

are using the fact that $\log a^b = b \log a$). Therefore taking $n_0 = C = 1$ satisfies the definition of big-O, and $\log_2 n! = O(n \log n)$.

Now we show $\log_2 n! = \Omega(n \log n)$. Assume without loss of generality that n is even. In the definition of $n!$, we'll use the first $n/2$ terms in the product to lower bound it:

$$n! \quad = \quad n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 2 \cdot 1 \quad \geq \quad \underbrace{\frac{n}{2} \cdot \frac{n}{2} \cdot \cdots \cdot \frac{n}{2}}_{n/2 \text{ times}} \quad = \quad \left(\frac{n}{2}\right)^{\frac{n}{2}}.$$

Taking the $\log$ of both sides gives us $\log_2 n! \geq \frac{n}{2} \log_2 \frac{n}{2}$.

Claim: For $n \geq 4$, $\frac{n}{2} \log_2 \frac{n}{2} \geq \frac{n}{4} \log_2 n$.

The proof of the claim is not difficult (some algebraic manipulation) and is left for the reader. Using the claim, we know that for $n \geq 4$, $\log_2 n! \geq \frac{n}{4} \log_2 n$. Therefore, taking $n_0 = 4$ and $c = 1/4$ satisfies the definition of big-Omega, and $\log_2 n! = \Omega(n \log n)$. ■

Exercise 6.4 (Composing polynomial time algorithms)**.**
Suppose that we have an algorithm A that runs another algorithm A' once as a subroutine. We know that the running time of A' is $O(n^k)$, $k \geq 1$, and the work done by A is $O(n^t)$, $t \geq 1$, if we ignore the subroutine A' (i.e., we don't count the steps taken by A'). What kind of upper bound can we give for the total running-time of A (which includes the work done by A')?

Solution. Let n be the length of the input to algorithm A. The total work done by A is $f(n) + g(n)$, where $g(n)$ is the work done by the subroutine A' and $f(n) = O(n^t)$ is the work done ignoring the subroutine A'.

We analyze $g(n)$ as follows. Note that in time $O(n^t)$, A can produce a string of length cn^t (for some constant c), and feed this string to A'. The running time of A' is $O(m^k)$, where m is the length of the input for A'. When A is run and calls A', the length of the input to A' can be $m = cn^t$. Therefore, the work done by A' inside A is $O(m^k) = O((cn^t)^k) = O(c^k n^{tk}) = O(n^{tk})$.

Since $f(n) = O(n^t)$ and $g(n) = O(n^{tk})$, we have $f(n) + g(n) = O(n^{tk})$. ■

Exercise 6.5 (TM complexity of $\{0^k 1^k : k \in \mathbb{N}\}$)**.**
In the TM model, a *step* corresponds to one application of the transition function. Show that $L = \{0^k 1^k : k \in \mathbb{N}\}$ can be decided by a TM in time $O(n \log n)$. Is this statement directly implied by Proposition (Intrinsic complexity of $\{0^k 1^k : k \in \mathbb{N}\}$)?

Solution. First of all, the statement is not directly implied by Proposition (Intrinsic complexity of $\{0^k 1^k : k \in \mathbb{N}\}$) because that proposition is about the RAM model whereas this question is about the TM model.

We now sketch the solution. Below is a medium-level description of a TM deciding the language $\{0^k 1^k : k \in \mathbb{N}\}$.

```
Repeat while both 0s and 1s remain on the tape:
    Scan the tape.
    If (# of 1s + # of 0s) is odd, reject.
    Scan the tape.
    Cross off every other 0 starting with first 0.
    Cross off every other 1 starting with first 1.
If no 0s and no 1s remain accept.
Else, reject.
```

Let n be the input length. Observe that in each iteration of the loop, we do $O(n)$ work, and the number of iterations is $O(\log n)$ because in each iteration, half of the non-crossed portion of the input gets crossed off (i.e. in each iteration, the number of 0's and 1's is halved). So the total running time is $O(n \log n)$. ■

Exercise 6.6 (Is polynomial time decidability closed under concatenation?)**.**
Assume the languages L_1 and L_2 are decidable in polynomial time. Prove or give a counter-example: $L_1 L_2$ is decidable in polynomial time.

Solution. Let M_1 be a decider for L_1 with running-time $O(n^k)$ and let M_2 be a decider for L_2 with running-time $O(n^t)$. We construct a polynomial-time decider for $L_1 L_2$ as follows:

```
On input x:
    For each of the |x| + 1 ways to divide x as yz:
        Run M1(y)
        If M1 accepts:
            Run M2(z)
            If M2 accepts, accept
    Reject
```

The input length is $n = |x|$. The for-loop repeats $n + 1$ times. In each iteration of the loop, we do at most $cn^k + c'n^t + c''$ work, where c, c', c'' are constants independent of n. So the total running-time is $O(n^{\max\{k,t\}+1})$. ■

Exercise 6.7 (Running time of the factoring problem)**.**
Consider the following problem: Given as input a positive integer N, output a non-trivial factor[1] of N if one exists, and output False otherwise. Give a lower bound using the $\Omega(\cdot)$ notation for the running-time of the following algorithm solving the problem:

> N: natural number.
> Non-Trivial-Factor($\langle N \rangle$):
> 1 For $i = 2$ to $N - 1$:
> 2 If $N\%i == 0$: Return i.
> 3 Return False.

Solution. The input is a number N, so the length of the input is n, which is about $\log_2 N$. In other words, N is about 2^n. In the worst-case, N is a prime number, which would force the algorithm to repeat $N - 2$ times. Therefore the running-time of the algorithm is $\Omega(N)$. Writing N in terms of n, the input length, we get that the running-time is $\Omega(2^n)$. ■

Exercise 6.8 (251st root)**.**
Consider the following computational problem. Given as input a number $A \in \mathbb{N}$, output $\lfloor A^{1/251} \rfloor$. Determine whether this problem can be computed in worst-case polynomial-time, i.e. $O(n^k)$ time for some constant k, where n denotes the number of bits in the binary representation of the input A. If you think the problem can be solved in polynomial time, give an algorithm in pseudocode, explain briefly why it gives the correct answer, and argue carefully why the running time is polynomial. If you think the problem cannot be solved in polynomial time, then provide a proof.

Solution. First note that the following algorithm, although correct, is exponential time.

[1] A non-trivial factor is a factor that is not equal to 1 or the number itself.

```
A: natural number.
Linear-Search(⟨A⟩):
1 For B = 0 to A:
2     If B^251 > A: Return B − 1.
```

The length of the input is n, which is about $\log_2 A$. In other words, A is about 2^n. The for loop above will repeat $\lfloor A^{1/251} \rfloor$ many times, so the running-time is $\Omega(A^{1/251})$, and in terms of n, this is $\Omega(2^{n/251})$.

To turn the above idea into a polynomial-time algorithm, we need to use binary search instead of linear search.

```
A: natural number.
Binary-Search(⟨A⟩):
1 lo = 0.
2 hi = A.
3 While (lo < hi):
4     B = ⌈(lo+hi)/2⌉.
5     If B^251 > A: hi = B − 1.
6     Else lo = B.
7 Return lo.
```

Since we are using binary search, we know that the loop repeats $O(\log A)$ times, or using n as our parameter, $O(n)$ times. All the variables hold values that are at most A, so they are at most n-bits long. This means all the arithmetic operations (plus, minus, and division by 2) in the loop can be done in linear time. Computing B^{251} is polynomial-time because we can compute it by doing integer multiplication a constant number of times, and the numbers involved in these multiplications are $O(n)$-bits long. Thus, the total work done is polynomial in n. ■

Chapter 7

The Science of Cutting Cake

Exercise 7.1 (Practice with cutting cake)**.**
Design a cake cutting algorithm for a set of players $N = \{1, \ldots, n\}$ that finds an allocation A with the property that there exists a permutation/bijection $\pi_A : N \to N$ such that for all $i \in N$, $V_i(A_i) \geq \frac{1}{2^{\pi(i)}}$. In words, there is an order on the players such that the first player has value at least $1/2$ for her piece, the second player has value at least $1/4$, and so on. The complexity of your algorithm in the Robertson-Webb model should be $O(n^2)$.

Solution. We can modify the Dubins-Spanier algorithm to solve this exercise question. The referee first makes n queries: $\text{Cut}_i(0, 1/2)$ for all i. She computes the minimum among these values, which we'll denote by y. Let's assume j is the player that corresponds to the minimum value. Then the referee assigns $A_j = [0, y]$. So player j gets a piece that she values at $1/2$. After this, we remove player j, and repeat the process on the remaining cake. So in the next stage, the referee makes $n - 1$ queries, $\text{Cut}_i(y, 1/4)$ for $i \neq j$, figures out the player corresponding to the minimum value, and assigns her the corresponding piece of the cake, which she values at $1/4$. This repeats until there is one player left. The last player gets the piece that is left.

The analysis of the running time of this algorithm is exactly the same as in the original Dubins-Spanier algorithm.

We need to show that the allocation produced by the algorithm is such that the first player (in the order the algorithms picks players) has value at least $1/2$ for her piece, the second player has value at least $1/4$ for her piece, and so on. This pretty much follows from the way the algorithm is set up and how allocations are made. The only things we need to check are:

(i) the queries never return "None",

(ii) the last player, call it ℓ, gets A_ℓ such that $V_\ell(A_\ell) \geq 1/2^n$.

To show (i), assume we have just completed iteration k, where $k \in \{1, 2, \ldots, n-1\}$. Let j be an arbitrary player who has not been removed yet. The important observation is that the piece of cake remaining at this point has value at least $1/2^k$ to player j (take a moment to verify this). Since this is true for any $k \in \{1, 2, \ldots, n-1\}$ and any player j that remains after iteration k, the queries never return "None". Part (ii) actually follows from the same argument. The cake remaining after iteration $n-1$ has value at least $1/2^{n-1}$ for the last player (which is indeed better than $1/2^n$. This completes the proof. ■

Exercise 7.2 (Finding an envy-free allocation)**.**
We say that a valuation function V is *piecewise constant* if there are points $x_1, x_2, \ldots, x_k \in [0, 1]$ such that $0 = x_1 < x_2 < \cdots < x_k = 1$ and for each $i \in \{1, 2, \ldots, k-1\}$, $V([x_i, x_{i+1}])$ is uniformly distributed over $[x_i, x_{i+1}]$.[1] Suppose we have n players such that each player has a piecewise constant valuation function. Show that in this case, an envy-free allocation always exists.

Solution. We sketch the idea, but do not prove the correctness. Each player's valuation function is associated with a set of points. Make a mark for each such point on the cake $[0, 1]$. The subinterval between any two adjacent marks x and y is such that for all players i, $V_i([x, y])$ is uniformly distributed. Divide each such subinterval into n pieces of equal length, and give one to each player. ■

[1] Uniformly distributed means that if we were to take any subinterval I of $[x_i, x_{i+1}]$ whose density/size is α fraction of the density/size of $[x_i, x_{i+1}]$, then $V(I) = \alpha \cdot V([x_i, x_{i+1}])$.

Chapter 8

Introduction to Graph Theory

Exercise 8.1 (Max number of edges in a graph)**.**
In an n-vertex graph, what is the maximum possible value for the number of edges in terms of n?

Solution. An edge is a subset of V of size 2, and there are at most $\binom{n}{2}$ possible subsets of size 2. ■

Exercise 8.2 (Application of Handshake Theorem)**.**
Is it possible to have a party with 251 people in which everyone knows exactly 5 other people in the party?

Solution. Create a vertex for each person in the party, and put an edge between two people if they know each other. Note that the question is asking whether there can be a 5-regular graph with 251 nodes. We use Theorem (Handshake Theorem) to answer this question. If such a graph exists, then the sum of the degrees would be 5×251, which is an odd number. However, this number must equal $2m$ (where m is the number of edges), and $2m$ is an even number. So we conclude that there cannot be a party with 251 people in which everyone knows exactly 5 other people. ■

Exercise 8.3 (Equivalent definitions of a tree)**.**
Show that if a graph has two of the properties listed in Definition (Tree, leaf, internal node), then it automatically has the third as well.

Solution. If a graph is connected and satisfies $m = n-1$, then it must be acyclic by Theorem (Min number of edges to connect a graph). If a graph is connected and acyclic, then it must satisfy $m = n - 1$, also by Theorem (Min number of edges to connect a graph). So all we really need to prove is if a graph is acyclic and satisfies $m = n - 1$, then it is connected. For this we look into the proof of Theorem (Min number of edges to connect a graph). If the graph is acyclic, this means that *every time* we put back an edge, we put one that satisfies (i) (here "(i)" is referring to the item in the proof of Theorem (Min number of edges to connect a graph)). This is because any edge that satisfies (ii) creates a cycle. Every time we put an edge satisfying (i), we reduce the number of connected components by 1. Since $m = n - 1$, we put back $n - 1$ edges. This means we start with n connected components (n isolated vertices), and end up with 1 connected component once all the edges are added back. So the graph is connected. ■

Exercise 8.4 (A tree has at least 2 leaves)**.**
Let T be a tree with at least 2 vertices. Show that T must have at least 2 leaves.

Solution. We use Theorem (Handshake Theorem) to prove this (i.e. $\sum_v \deg(v) = 2m$). If a tree has less than 2 leaves, then the sum of the degrees of the vertices would be *at least*

$$1 + 2(n-1) = 2n - 1$$

(in the worst-case, we have 1 leaf and $n - 1$ vertices with degree 2). This value must equal $2m$, which is always equal to $2(n-1) = 2n - 2$ in a tree. This is a contradiction since $2n - 1 > 2n - 2$. ■

Exercise 8.5 (Max degree is at most number of leaves)**.**
Let T be a tree with L leaves. Let Δ be the largest degree of any vertex in T. Prove that $\Delta \leq L$.

Solution. It is instructive to read all 3 proofs.
Proof 1: We use Theorem (Handshake Theorem). The degree sum in a tree is always $2n-2$ since $m = n-1$. Let v be the vertex with maximum degree Δ. The vertices that are not v or leaves must have degree at least 2 each, so the degree sum is *at least* $\deg(v) + L + 2(n-L-1)$. So we must have $2n-2 \geq \deg(v) + L + 2(n-L-1)$, which simplifies to $L \geq \deg(v) = \Delta$, as desired.
Proof 2: We induct on the number of vertices. For $n \leq 3$, this follows by inspecting the unique tree on n vertices. For $n > 3$, pick an arbitrary leaf u and delete it (and all the edges incident to u). Let $T-u$ denote this graph, which is a tree (it is connected and acyclic). Also, we let $L(T)$ denote the number of leaves in T and $L(T-u)$ to denote the number of leaves in $T-u$. We make similar definitions for $\Delta(T)$ and $\Delta(T-u)$ regarding the maximum degrees. Note that $L(T) \geq L(T-u)$. There are two cases to consider:

1. $\Delta(T-u) = \Delta(T)$
2. $\Delta(T-u) = \Delta(T) - 1$

If case 1 happens, then by the induction hypothesis $L(T-u) \geq \Delta(T-u) = \Delta(T)$. But this implies $L(T) \geq \Delta(T)$ (since $L(T) \geq L(T-u)$), as desired.

Let v be the neighbor of u. If case 2 happens, then v is the only vertex of maximum degree in T. In particular, v cannot be a leaf in $T-u$. So $L(T) = L(T-u)+1$. The induction hypothesis yields $L(T-u) \geq \Delta(T-u) = \Delta(T)-1$. Combining this with $L(T) = L(T-u)+1$ we get $L(T) \geq \Delta(T)$, as desired.
Proof 3: Let v be a vertex in the tree such that $\deg(v) = \Delta$. Consider the graph $T-v$ obtained by deleting v and all the edges incident to it. Since T is a tree, we know that $T-v$ contains Δ connected components; let us denote them $T_1, \ldots, T_\Delta$. Since T is acyclic, each of the T_i's are also acyclic. Since each T_i is connected and acyclic, each one is a tree. There are two possibilities for each T_i:

(i) T_i consists of a single vertex. Then that vertex is a leaf in T.

(ii) T_i is not a single vertex, and so has at least 2 leaves (by Exercise (A tree has at least 2 leaves)). At least one of these leaves is not connected to v and therefore must be a leaf in T.

In either case, one vertex in T_i is a leaf in T. This is true for all $T_1, \ldots, T_\Delta$. Hence we have at least Δ leaves in T. ■

Exercise 8.6 (MST with negative costs)**.**
Suppose an instance of the Minimum Spanning Tree problem is allowed to have negative costs for the edges. Explain whether we can use the Jarník-Prim algorithm to compute the minimum spanning tree in this case.

Solution. Yes, we can. Assign a rank to each edge of the graph based on its cost: the highest cost edge gets the highest rank and the lowest cost edge gets the lowest rank. When making its decisions, the Jarník-Prim algorithm only cares about the ranks of the edges, and not the specific costs of the edges. The algorithm would output the same tree even if we add a constant C to the costs of all the edges since this would not change the rank of the edges. And indeed, adding a constant to the cost of each edge does not change what the minimum spanning tree is. Hence, we can turn any instance with negative costs into and equivalent one with non-negative costs by adding a large enough constant to all the edges without changing the tree that is output.
(Note: In fact the original algorithm would output the minimum cost spanning tree even if the edge costs are allowed to be negative. There is not even a need to add a constant to the edge costs.) ■

Exercise 8.7 (Maximum spanning tree)**.**
Consider the problem of computing the maximum spanning tree, i.e., a spanning tree that maximizes the sum of the edge costs. Explain whether the Jarník-Prim algorithm solves this problem if we modify it so that at each iteration, the algorithm chooses the edge between V' and $V \backslash V'$ with the maximum cost.

Solution. Let (G, c) be the input, where $G = (V, E)$ is a graph and $c : E \to \mathbb{R}^+$ is the cost function. Let $c' : E \to \mathbb{R}^-$ be defined as follows: for all $e \in E$, $c'(e) = -c(e)$. Let $A_{\min}$ be the original Jarník-Prim algorithm and let $A_{\max}$ be the Jarník-Prim algorithm where we pick the maximum cost edge in each iteration. There are a couple of important observations:

1. The minimum spanning tree for (G, c') is the maximum spanning tree for (G, c).
2. Running $A_{\max}(G, c)$ is equivalent to running $A_{\min}(G, c')$, and they output the same spanning tree.

From Exercise (MST with negative costs), we know $A_{\min}(G, c')$ gives us a minimum cost spanning tree. So $A_{\max}(G, c)$ gives the correct maximum cost spanning tree. ■

Exercise 8.8 (Kruskal's algorithm)**.**
Consider the following algorithm for the MST problem (which is known as Kruskal's algorithm). Start with MST being the empty set. Go through all the edges of the graph one by one from the cheapest to the most expensive. Add the edge to the MST if it does not create a cycle. Show that this algorithm correctly outputs the MST.

Solution. We do not have the solution to this problem at this time. ■

Exercise 8.9 (Cycle implies no topological order)**.**
Show that if a directed graph has a cycle, then it does not have a topological order.

Solution. Let $G = (V, A)$ be a directed graph and suppose $u_1, u_2, \ldots, u_k, u_1$ is a cycle in G. This means that for all $i \in \{1, 2, \ldots, k-1\}$, $(u_i, u_{i+1}) \in A$, and $(u_k, u_1) \in A$. If there is a topological order f of G, then by definition, it must be the case that

$$f(u_1) < f(u_2) < f(u_3) < \cdots < f(u_k) < f(u_1).$$

This implies $f(u_1) < f(u_k) < f(u_1)$, which is impossible. ■

Exercise 8.10 (Topological sort, correctness of naïve algorithm)**.**
Show the algorithm above correctly solves the topological sorting problem, i.e., show that for $(u, v) \in A$, $f(u) < f(v)$. What is the running time of this algorithm?

Solution. We will use the following observation: if an algorithm removes an edge $(u, v) \in A$, then it must be because v is chosen as a sink vertex and removed from the graph.

We now prove that the algorithm is correct by a proof by contradiction. Suppose $(u, v) \in A$ such that $f(u) > f(v)$. This means that u was removed from the graph before v was removed. At the moment that u is removed, u must be a sink (i.e. it must not have any outgoing edges). This implies the edge (u, v) must have been removed at a previous iteration. But the only way

(u, v) would be removed is if v was chosen to be a sink vertex and removed. This implies that v must have been removed before u, which is the desired contradiction.

A straightforward implementation of the algorithm would result in a running time of at least $\Omega(n^2)$ since the algorithm has n iterations, and in each iteration, a sink vertex must be found and removed (which takes $\Omega(n)$ steps). ■

Chapter 9

Matchings in Graphs

Exercise 9.1 (Number of perfect matchings in a complete graph)**.**
Let n be even, and let G be the complete graph[1] on n vertices. How many different perfect matchings does G contain?

Solution. The answer is $(n-1)(n-3)\cdots 1$. We leave it to the reader to verify this. ■

Exercise 9.2 (Graphs with max degree at most 2)**.**
Let $G = (V, E)$ be a graph such that all vertices have degree at most 2. Then prove that G consists of disjoint paths and cycles (where we count an isolated vertex as a path of length 0).

Solution. Consider a graph G such that all vertices have degree at most 2. We want to show that it consists of disjoint paths and cycles. We prove this by induction on the number of vertices.

Pick an arbitrary vertex v in the graph. Removing v results in a graph $G - v$ such that every vertex has degree at most 2. Since $G - v$ has one less vertex, by induction hypothesis, $G - v$ consists of disjoint paths and cycles. There are 3 cases to consider: $\deg(v) = 0, 1$, or 2. It is not hard to see that in each case, adding v back to the graph preserves the property that the graph is a collection of disjoint paths and cycles. (Verify this part for yourself.) ■

Exercise 9.3 (A tree can have at most one perfect matching)**.**
Show that a tree can have at most one perfect matching.

Solution. The proof is by contradiction, so suppose a tree has two different perfect matchings M and M'. Let S be the symmetric difference between M and M', i.e., $S = (M \cup M') \backslash (M \cap M')$. Since $M \neq M'$, $|S| > 1$. The set S corresponds to a graph in which each vertex has degree at most 2. So the graph consists of disjoint paths and cycles. But it cannot contain any cycles since trees are acyclic. It also cannot contain a path. This is because the existence of a degree 1 vertex in S implies that this vertex is not covered/matched by either M or M' (verify this yourself), and this would contradict the fact that M and M' are *perfect* matchings covering all vertices. So S must be the empty set, which contradicts our assumption that $|S| > 1$. ■

Exercise 9.4 (Practice with perfect matchings)**.**

(a) Let G be a bipartite graph on $2n$ vertices such that every vertex has degree at least n. Prove that G must contain a perfect matching.

(b) Let $G = (X, Y, E)$ be a bipartite graph with $|X| = |Y|$. Show that if G is connected and every vertex has degree at most 2, then G must contain a perfect matching.

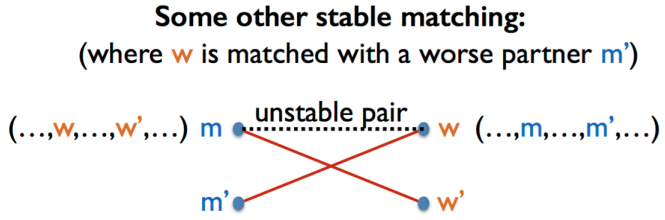
Solution. Part a: If every vertex has degree n, it must be the case that the graph is $G = (X, Y, E)$ where $|X| = |Y| = n$. It must also be the case that the graph is a complete bipartite graph (i.e., every possible edge is present). So clearly Hall's theorem applies and the graph has a perfect matching.

Part b: A graph with max degree at most 2 consists of disjoint paths and cycles (Exercise (Graphs with max degree at most 2)). Since the graph is connected, it must consist of a single path or a single cycle containing all the vertices. In either case, it is not hard to see that the graph must contain a perfect matching: If the graph is a path, then we can take every other edge (starting with the first edge) along the path to form a perfect matching. If the graph consists of a cycle, it has two different perfect matchings. ■

[1] A complete graph is a graph in which every possible edge is present.

Exercise 9.5 (Gale-Shapley is female pessimal)**.**
Show that the Gale-Shapley algorithm always matches a female $w \in Y$ with its worst valid partner, i.e., it returns $\{(\text{worst}(w), w) : w \in Y\}$.

Solution. The proof is by contradiction, so suppose that the Gale-Shapley algorithm returns a matching in which some w is matched to m, but m is not the worst valid partner of w. Let m' be the worst valid partner of w. So there is some other stable matching in which m' and w are matched. Let w' be the match of m in this stable matching.



We claim that in this stable matching (m, w) forms an unstable pair. First, w prefers m over m' because m' is the worst valid partner of w. Second, m prefers w over w' because the Gale-Shapley algorithm matches m and w, and so w must be the best valid partner of m by Theorem (Gale-Shapley is male optimal). ■

Exercise 9.6 (Is there a unique stable matching?)**.**
Give a polynomial time algorithm that determines if a given instance of the stable matching problem has a unique solution or not.

Solution. Run the Gale-Shapley algorithm A with men proposing to women. Run the algorithm A' by reversing the roles of men and women so that women propose to men. We claim there is a unique stable matching if and only if both A and A' output the same stable matching.

One direction is clear: if there is a unique stable matching, than A and A' must return the same matching. For the other direction, suppose A and A' return the same stable matching. We know that (i) A returns a male-optimal matching (Theorem (Gale-Shapley is male optimal)); (ii) A' returns a male-pessimal matching (Exercise (Gale-Shapley is female pessimal)). Since A and A' return the same matching, we must have that for all males m, $\text{best}(m) = \text{worst}(m)$. And this implies that in every stable matching, every m is matched to $\text{best}(m) = \text{worst}(m)$. So there must be only one stable matching. ■

Exercise 9.7 (Identical preferences)**.**
Suppose we are given an instance of the stable matching problem in which all the men's preferences are identical to each other, and all the women's preferences are identical to each other. Prove or disprove: there is only one stable matching for such an instance.

Solution. There is a unique stable matching for such an instance. To see this, let m_i be the man ranked i'th by the women, and let w_i be the woman ranked i'th by the men. Then notice that in any stable matching, m_1 and w_1 must be matched to each other because otherwise they would form an unstable pair. Given that m_1 and w_1 must be matched, m_2 must be matched to w_2 since otherwise, they would form an unstable pair. Proceeding this way, we see that for all i, m_i must be matched to w_i. ■

Exercise 9.8 (Stable roommates problem)**.**
Consider the following variant of the stable matching problem. The input is a complete graph on n vertices (not necessarily bipartite), where n is even. Each vertex has a preference list over every other vertex in the graph. The goal is to find a stable matching. Give an example to show that a stable matching does not always exist.

Solution. Consider the example with 4 nodes a, b, c and d, and the following preference lists:
a: (c, b, d)
b: (a, c, d)
c: (b, a, d)
d: (a, c, b)
Notice that d is the last choice of a, b and c. On the other hand, in any stable matching, d has to be matched with someone. If d is matched with a, then (a, b) is an unstable pair. If d is matched with b, then (b, c) is an unstable pair. And if d is matched with c, then (a, c) is an unstable pair. In all cases, there is an unstable pair, so a stable matching does not exist. ■

Chapter 10

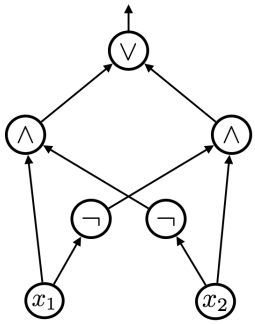
Boolean Circuits

Exercise 10.1 (Drawing circuits)**.**
Draw a circuit that computes the following functions.

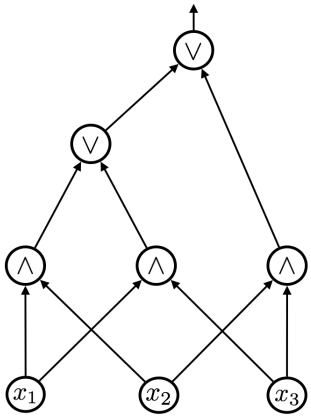
(a) The parity function PAR : $\{0,1\}^2 \to \{0,1\}$ on 2 variables, which is defined as PAR$(x_1, x_2) = 1$ iff $x_1 + x_2$ is odd.

(b) The majority function MAJ : $\{0,1\}^3 \to \{0,1\}$ on 3 variables, which is defined as MAJ$(x_1, x_2, x_3) = 1$ iff $x_1 + x_2 + x_3 \geq 2$.

Solution. Part (a):



Part (b):

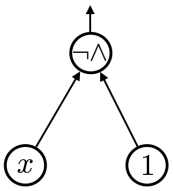


■

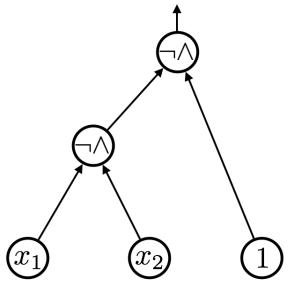
Exercise 10.2 (NAND is "universal")**.**
Define a NAND gate as NAND$(x, y) = \neg(x \wedge y)$. Show that any circuit with AND, OR and NOT gates can be converted into an equivalent circuit (i.e. a circuit computing the same function) that uses *only* NAND gates (in addition to the input gates and constant gates). The size of this circuit should be at most a constant times the size of the original circuit.

Solution. For this question, all we need to do is show how to compute OR, AND, and NOT just with NAND gates. Below are the constructions.

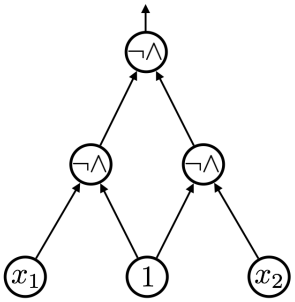
We can compute NOT of a variable x as follows:



We can compute AND of x_1 and x_2 by applying a NAND gate to x_1 and x_2, and then negate the output:



To compute OR of x_1 and x_2, we use the fact that $x_1 \vee x_2 = \neg(\neg x_1 \wedge \neg x_2)$:



Any circuit with AND, OR, and NOT gates can be converted to a one using only NAND gates by replacing each AND, OR, and NOT gate with the corresponding circuit as shown above. For each gate we replace, we need to use at most 3 NAND gates. So the size of our circuit with NAND gates will be at most 3 times as the size of the original circuit. ■

Exercise 10.3 (Circuit complexity of parity)**.**
Let $L \subseteq \{0,1\}^*$ be the set of words which contain an odd number of 1's. Show that the circuit complexity of L is $\Theta(n)$.

Solution. Note that the decision problem $f : \{0,1\}^* \to \{0,1\}$ corresponding to L is simply the *parity* function: $f(x) = x_1 + x_2 + \cdots + x_n \mod 2$, where $n = |x|$. Write $f = (f^0, f^1, f^2, \ldots)$ as in Note (Dividing the set of words by length).

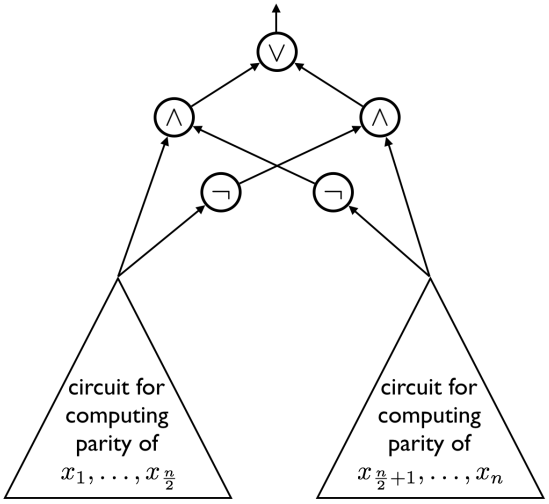
The lower bound of $\Omega(n)$ on the circuit complexity is relatively straightforward. To compute f^n, the circuit needs to access all the input bits. This is because if it does not access x_i for some i, the circuit would output the same bit whether $x_i = 0$ or $x_i = 1$. However, a correct circuit computing the parity function cannot have this behavior. The parity of the sum of the bits depends on all of the input bits. Since the circuit must make use of all of the input gates x_1 to x_n, and we count the input gates when computing the size of the circuit, we can conclude that the circuit complexity of f is $\Omega(n)$. (In fact, even if we did not count the input gates when computing the size of a circuit, we could still conclude that the circuit must have size $\Omega(n)$. Can you see why?)

For the upper bound, we present an explicit circuit construction that computes the parity of the input bits. The parity of a single input bit is easily computed as follows:

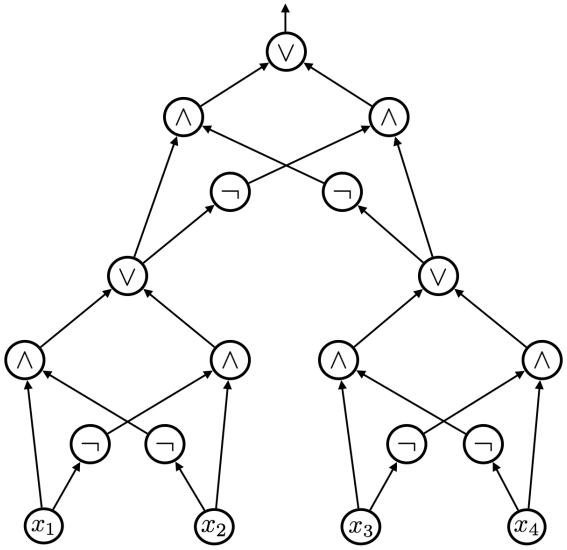
Exercise (Drawing circuits), part (a), shows us how to do parity of two bits. For simplicity, assume n is a power of 2. Then note that the parity of $x_1, x_2, \ldots, x_n$, denoted by $\text{PAR}_n(x_1, \ldots, x_n)$, can be written as

$$\text{PAR}_n(x_1, \ldots, x_n) = \text{PAR}_2(\text{PAR}_{\frac{n}{2}}(x_1, \ldots, x_{\frac{n}{2}}), \text{PAR}_{\frac{n}{2}}(x_{\frac{n}{2}+1}, \ldots, x_n)).$$

This gives us a recursive way to construct the circuit computing parity. The recursive construction is illustrated below:



Note that the gadget above the two triangles is just a circuit computing the parity of two bits. For $n = 4$, the construction would materialize as follows:



If $S(n)$ is the size of the above circuit for computing parity over n input variables, then $S(n) = 2S(n/2) + 5$, and $S(1) = 1$. Solving this recursion, we get $S(n) = O(n)$, as desired.

(We leave it to the reader to handle the case when n is not a power of 2.) ■

Exercise 10.4 ($O(n2^n)$ upper bound on circuit complexity)**.**
Show that any language L can be computed by a circuit family of size $O(n2^n)$.

Solution. We will not give a complete solution to this problem, but give enough of a hint that hopefully you will be able to figure out the complete argument yourself. Let's take an example function with $n = 3$ variables.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Then we claim that the following Boolean formula correctly represents f:

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

First, try to see how we came up with this Boolean formula, and why it correctly represents the function. Once you understand the construction, this will give you a way of constructing a formula for *any* Boolean function. Note that a Boolean formula can be represented as a Boolean circuit. What is the maximum number of gates the circuit would have for any function? ■

Chapter 11

Polynomial-Time Reductions

Exercise 11.1 (Transitivity of Karp reductions)**.**
Show that if $A \leq_m^P B$ and $B \leq_m^P C$, then $A \leq_m^P C$.

Solution. Suppose $A \leq_m^P B$ and $B \leq_m^P C$. Let $f : \Sigma^* \to \Sigma^*$ be the map that establishes $A \leq_m^P B$ and let $g : \Sigma^* \to \Sigma^*$ be the map that establishes $B \leq_m^P C$. So $x \in A$ if and only if $f(x) \in B$. And $x \in B$ if and only if $g(x) \in C$. Both f and g are computable in polynomial time.

To show $A \leq_m^P C$, we define $h : \Sigma^* \to \Sigma^*$ such that $h = g \circ f$. That is, for all $x \in \Sigma^*$, $h(x) = g(f(x))$. We need to show that

- $x \in A$ if and only if $h(x) \in C$;
- h is computable in polynomial time.

For the first part, note that by the properties of f and g, $x \in A$ if and only if $f(x) \in B$ if and only if $g(f(x)) \in C$ (i.e., $h(x) \in C$).
(If you are having trouble following this, you can break this part up into two parts:
(i) $x \in A \implies h(x) \in C$, (ii) $x \notin A \implies h(x) \notin C$.)
For the second part, if f is computable in time $O(n^k)$, $k \geq 1$, and g is computable in time $O(n^t)$, $t \geq 1$, then h can be computed in time $O(n^{kt})$. (Why?) ■

Exercise 11.2 (IS reduces to CLIQUE)**.**
How can you modify the above reduction to show that IS $\leq_m^P$ CLIQUE?

Solution. We can use exactly the same reduction as the one in the reduction from CLIQUE to IS:

$G = (V, E)$: graph. k: positive integer.
$f(\langle G, k\rangle)$:
1 $E' = \{\{u, v\} : \{u, v\} \notin E\}$.
2 Output $\langle G' = (V, E'), k\rangle$.

This reduction establishes IS $\leq_m^P$ CLIQUE. The proof of correctness is the same as in the proof of Theorem (CLIQUE reduces to IS); just interchange the words "clique" and "independent set" in the proof. ■

Exercise 11.3 (Hamiltonian path reductions)**.**
Let $G = (V, E)$ be a graph. A *Hamiltonian path* in G is a path that visits every vertex of the graph exactly once. The HAMPATH problem is the following: given a graph $G = (V, E)$, output True if it contains a Hamiltonian path, and output False otherwise.

(a) Let $L = \{\langle G, k\rangle : G$ is a graph, $k \in \mathbb{N}$, G has a path of length $k\}$. Show that HAMPATH $\leq_m^P L$.

(b) Let $K = \{\langle G, k\rangle : G$ is a graph, $k \in \mathbb{N}$, G has a spanning tree with $\leq k$ leaves$\}$. Show that HAMPATH $\leq_m^P K$.

Solution. Part (a): We need to show that there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ HAMPATH if and only if $f(x) \in L$. Below we present f.

$G = (V, E)$: graph.
$f(\langle G\rangle)$:
1 Output $\langle G, |V| - 1\rangle$.

We first prove the correctness of the reduction. If $x \in$ HAMPATH, then x corresponds to an encoding of a graph G that contains a Hamiltonian path. Let n be the number of vertices in the graph. A Hamiltonian path visits every vertex of the graph exactly once, so has length $n-1$ (the length of a path is the number of edges along the path). Therefore, by the definition of L, we must have $f(x) = \langle G, n-1 \rangle \in L$. For the converse, suppose $f(x) \in L$. Then it must be the case that $f(x) = \langle G, n-1 \rangle$, where $x = \langle G \rangle$, G is some graph, and n is the number of vertices in that graph. Furthermore, by the definition of L, it must be the case that G contains a path of length $n-1$. A path cannot repeat any vertices, so this path must be a path visiting every vertex in the graph, that is, it must be a Hamiltonian path. So $x \in$ HAMPATH.

To see that the reduction is polynomial time, note that the number of vertices in the given graph can computed in polynomial time. So the function f can be computed in polynomial time.

Part (b): We need to show that there is a poly-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ HAMPATH if and only if $f(x) \in K$. Below we present f.

```
G = (V, E): graph.
f(⟨G⟩):
1 Output ⟨G, 2⟩.
```

A note: For the correctness proof below, we are going to ignore the edge case where the graph has 1 vertex. By convention, we don't allow graphs with 0 vertices.

We now prove the correctness of the reduction. If $x \in$ HAMPATH, then x corresponds to an encoding of a graph G that contains a Hamiltonian path. A Hamiltonian path visits every vertex of the graph, so it forms a spanning tree with 2 leaves. Therefore, by the definition of L, we must have $f(x) = \langle G, 2 \rangle \in L$. For the converse, suppose $f(x) \in L$. Then it must be the case that $f(x) = \langle G, 2 \rangle$, where $x = \langle G \rangle$ and G is some graph. Furthermore, by the definition of L, G must contain a spanning tree with 2 leaves (recall that every tree with at least 2 vertices must contain at least 2 leaves). A tree with exactly 2 leaves must be a path (prove this as an exercise). Since this path is a spanning tree, it must contain all the vertices. Therefore this path is a Hamiltonian path in G. So $x \in$ HAMPATH.

It is clear that the function f can be computed in polynomial time. This completes the proof. ■

Chapter 12

Non-Deterministic Polynomial Time

Exercise 12.1 (CLIQUE is in NP)**.**
Show that CLIQUE $\in$ NP.

Solution. To show CLIQUE is in NP, we need to show that there is a polynomial-time verifier A with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP). We start by presenting A. (We are using A to denote the verifier and not V because we will use V to denote the vertex set of a graph.)

$G = (V, E)$: graph. k: positive integer.
$A(\langle G, k\rangle, u)$:
1 If u does not correspond to an encoding of a set $S \subseteq V$ with $|S| = k$, reject.
2 For each pair of vertices $\{u, v\}$ in S:
3 If $\{u, v\} \notin E$, reject.
4 Accept.

We first show that the verifier A satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means that x corresponds to a valid encoding of a graph $G = (V, E)$ together with a number $k \in \mathbb{N}^+$. Furthermore, it must be the case that there is some $S \subseteq V$ with $|S| = k$ such that S forms a k-clique. When u is the encoding of such an S, then $|u| = O(n)$ where n is the length of x, and the verifier A accepts (x, u). On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a graph $G = (V, E)$ together with a number $k \in \mathbb{N}^+$ or (ii) it is a valid encoding $\langle G, k\rangle$, but there is no $S \subseteq V$, $|S| = k$, that forms a k-clique. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a set $S \subseteq V$ with $|S| = k$ makes the verifier reject. Furthermore, any u that does correspond to such an S cannot form a k-clique. In this case, the for loop will detect this and the verifier again rejects, as desired.

Now we show the verifier is polynomial-time. Checking whether u is a valid encoding of a set $S \subseteq V$ with $|S| = k$ is polynomial time. If this check passes, then the for-loop repeats at most $O(|V|^2)$ many times, and the body of the loop can be carried out in polynomial time. So in total, the work being done is polynomial time. ■

Exercise 12.2 (IS is in NP)**.**
Show that IS $\in$ NP.

Solution. The argument is very similar to the one above. Here is the verifier:

$G = (V, E)$: graph. k: positive integer.
$A(\langle G, k\rangle, u)$:
1 If u does not correspond to an encoding of a set $S \subseteq V$ with $|S| = k$, reject.
2 For each pair of vertices $\{u, v\}$ in S:
3 If $\{u, v\} \in E$, reject.
4 Accept.

We skip the arguments for correctness and running time. ■

Exercise 12.3 (3SAT is in NP)**.**
Show that 3SAT $\in$ NP.

Solution. To show 3SAT is in NP, we need to show that there is a polynomial-time verifier V with the properties stated in Definition (Non-deterministic polynomial time, complexity class NP). We start by presenting V.

> F: 3CNF formula.
> $V(\langle F \rangle, u)$:
> 1 If u does not correspond to a valid $0/1$ assignment to the variables in F, reject.
> 2 Compute the output of the formula $F(u)$.
> 3 If the output is 0, reject.
> 4 Else, accept.

We first show that the verifier V satisfies the two conditions stated in Definition (Non-deterministic polynomial time, complexity class NP). If x is in the language, that means that x corresponds to a valid encoding of a 3CNF formula. Furthermore, there is some $0/1$-assignment to the variables that makes the formula output 1. When u is this $0/1$-assignment, then $|u| = O(n)$ (where n is the length of x), and the verifier accepts the input (x, u). On the other hand, if x is not in the language, then either (i) x is not a valid encoding of a CNF formula in which every clause has exactly 3 literals or (ii) it is a valid encoding but the formula is not satisfiable. In case (i), the verifier rejects (which is done implicitly since the input is not of the correct type). In case (ii), any u that does not correspond to a $0/1$-assignment to the variables makes the verifier reject. Furthermore, any u that does correspond to a $0/1$-assignment to the variables must be such that, with this assignment, the formula evaluates to 0. Therefore, in this case, the verifier again rejects, as desired.

Now we show the verifier is polynomial-time. To check whether u is a valid $0/1$-assignment to the variables takes polynomial time since you just need to check that you are given t bits, where t is the number of variables. The output of the formula can be computed in polynomial time since it takes constant number of steps to compute each clause (and the number of clauses is bounded by the length of the input). ■

Exercise 12.4 (NP is contained in EXP)**.**
Show that NP $\subseteq$ EXP.

Solution. To show NP $\subseteq$ EXP, it suffices to argue that any $L \in$ NP is also in EXP. So take an arbitrary $L \in$ NP. By the definition of NP we know that there is some polynomial-time verifier TM V and constant $k > 0$ such that for all $x \in \Sigma^*$:

- if $x \in L$, then there is some $u \in \Sigma^*$ with $|u| \leq |x|^k$ such that $V(x, u)$ accepts;
- if $x \notin L$, then for all $u \in \Sigma^*$, $V(x, u)$ rejects.

We construct a decider A for L as follows.

> $A(x)$:
> 1 For all $u \in \Sigma^*$ with $|u| \leq |x|^k$ do:
> 2 Run $V(x, u)$ and if it accepts, accept.
> 3 Reject.

Correctness: If $x \in L$, then we know that for some $u \in \Sigma^*$ with $|u| \leq |x|^k$, $V(x, u)$ accepts. Therefore A accepts as well. If on the other hand $x \notin L$, then for all $u \in \Sigma^*$, $V(x, u)$ rejects. Therefore A rejects as well.

Running time: The running time of A is $O(2^{n^C})$ for an appropriately chosen constant C. We omit the details of this part. ■

Exercise 12.5 (MSAT is NP-complete)**.**
The MSAT problem is defined very similarly to SAT (for the definition of SAT, see Definition (Boolean satisfiability problem)). In SAT, we output True if and only if the input CNF formula has at least one satisfying truth assignment to the variables. In MSAT, we want to output True if and only if the input CNF formula has at least two distinct satisfying assignments.

Show that MSAT is NP-complete.

Solution. Showing MSAT is in NP: To show that MSAT is in NP, we need to show that there is a polynomial time verifier TM V and $t \in \mathbb{N}$ such that:

(i) if $x \in$ MSAT, then there is some u, $|u| \leq |x|^t$, such that $V(x, u)$ accepts;

(ii) if $x \notin$ MSAT, then for all u, $V(x, u)$ rejects.

We present the verifier.

> F: CNF formula.
> $V(\langle F \rangle, u)$:
> 1 If u is not a valid encoding of two distinct $0/1$ assignments to the input variables, reject.
> 2 If both assignments in u satisfy F, accept.
> 3 Else, reject.

The verifier is polynomial time: Checking whether u a valid encoding of two distinct $0/1$ assignments to the input variables can be done in polynomial time. Given a $0/1$ assignment to the input variables, we can plug those values into the input CNF formula, and evaluate the output of the formula. This can be done in polynomial time. Since every step can be done in polynomial time, the verifier has polynomial time complexity.

Correctness: If $x \in$ MSAT, then x must a valid encoding of a CNF formula F. Furthermore, F must be such that there are at least two distinct $0/1$ assignments to the input variables that make F evaluate to 1. When u is the encoding of such two distinct $0/1$ assignments, the verifier will accept as desired. Note that clearly for such a u, $|u| \leq O(|F|)$, i.e., the proof is polynomial in the length of F. If $x \notin$ MSAT, then either x is not a valid encoding of a CNF formula (in which case the verifier rejects, which is done implicitly since the input is not of the correct type), or x is a valid encoding of a CNF formula, but there is at most one $0/1$ assignment to the input variables that makes the formula output 1. In the latter case, if u is not the encoding of two distinct $0/1$ assignments, we reject. And if it is, then the last step detects that at least one of these $0/1$ assignments makes the formula output 0, and so the verifier rejects again, as desired.

Showing MSAT is NP-hard: To show that MSAT is NP-hard, we show a Karp reduction from SAT (which we know is NP-hard because we have shown 3SAT is NP-hard) to MSAT. To show the Karp reduction, we need to show that there is a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ SAT if and only if $f(x) \in$ MSAT. Here is the function.

> F: CNF formula.
> $f(\langle F \rangle)$:
> 1 Let z be a variable name not included in F.
> 2 Let $F' = F \wedge (z \vee \neg z)$.
> 3 Output $\langle F' \rangle$.

Polynomial time: It is pretty clear that each step above can be carried out in polynomial time (with respect to the length of F).

Correctness: Suppose $x \in$ SAT. Then $x = \langle F \rangle$ for some satisfiable CNF formula F. Let $y_1, \ldots, y_n$ be the variables in F. And suppose $y_1 = b_1, \ldots, y_n = b_n$ is a satisfying assignment for F, where $b_i \in \{0, 1\}$ for all i. Then observe that both $y_1 = b_1, \ldots, y_n = b_n, z = 0$ and $y_1 = b_1, \ldots, y_n = b_n, z = 1$ are satisfying assignments for F', and so $\langle F' \rangle \in$ MSAT. For the converse, assume $\langle F' \rangle \in$ MSAT. This means $F' = (F \wedge (z \vee \neg z)) = (F \wedge \text{True})$ is satisfiable, which implies F must satisfiable. So $x = \langle F \rangle \in$ SAT. ■

Exercise 12.6 (BIN is NP-complete)**.**
In the PARTITION problem, we are given n non-negative integers $a_1, a_2, \ldots, a_n$, and we want to output True if and only if there is a way to partition the integers into two parts so that the sum of the two parts are equal. In other words, we want to output True if and only if there is set $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$.

In the BIN problem we are given a set of n items with non-negative sizes $s_1, s_2, \ldots, s_n \in \mathbb{N}$ (not necessarily distinct), a capacity $C \geq 0$ (not necessarily an integer), and an integer $k \in \mathbb{N}$. We want to output True if and only if the items can be placed into at most k bins such that the total size of items in each bin does not exceed the capacity C.

Show that BIN is NP-complete assuming PARTITION is NP-hard.

Solution. Showing BIN is in NP: To show that BIN is in NP, we need to show that there is a polynomial time verifier TM V and $t \in \mathbb{N}$ such that:

(i) if $x \in$ BIN, then there is some u, $|u| \leq |x|^t$, such that $V(x, u)$ accepts;

(ii) if $x \notin$ BIN, then for all u, $V(x, u)$ rejects.

$s_1, \ldots, s_n, k$: non-negative integers. C: non-negative real.
$V(\langle s_1, s_2, \ldots, s_n, C, k \rangle, u)$:
1 If $k > n$, set $k = n$.
2 If u is not a valid encoding of a partition $(B_1, \ldots, B_k)$ of $\{1, \ldots, n\}$, reject.
3 If for some $j = 1, ..., k$, $\sum_{i \in B_j} s_i > C$, reject.
4 Else reject.

The verifier is polynomial time: (First note that the n in the question does not denote the input length. When we say "polynomial time", it is with respect to the input length, which is $|\langle s_1, s_2, \ldots, s_n, C, k \rangle| + |u|$.) Checking that u is a partition of $\{1, \ldots, n\}$ into k parts, we need to check that there are k sets, their union is $\{1, \ldots, n\}$, and that they are pair-wise disjoint. All of these can be easily checked in polynomial time. In the 3rd step, we implicitly have a loop that repeats $k \leq n$ times. In each iteration, we do at most n additions, which is polynomial time in the input length (even if the s_i's are super big, note that they are part of the input and contribute to the length; addition is polynomial time).

Correctness: If $x \in$ BIN, then first, it must be a valid encoding. Furthermore, there must be a partition $(B_1, \ldots, B_k)$ of $\{1, \ldots, n\}$ such that for each $j = 1, ..., k$, $\sum_{i \in B_j} s_i \leq C$ (this is true by the definition of the problem). When u represents such a partition, the verifier correctly accepts. Such a u has length polynomial in n, and therefore polynomial in the input length, as desired.

If $x \notin$ BIN, then either x is an invalid encoding (in which case the verifier rejects, which is done implicitly since the input is not of the correct type), or x is a valid encoding, but there is no way to partition the elements into k bins $(B_1, \ldots, B_k)$ such that $\sum_{i \in B_j} s_i \leq C$ for all bins. If u does not correspond to a

partition $(B_1, \ldots, B_k)$, then we reject. And if it does, we successfully reject on the 3rd step. This completes the correctness argument.

Showing BIN is NP-hard: To show that BIN is NP-hard, we show a Karp reduction from PARTITION to BIN. To show the Karp reduction, we need to show that there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in$ PARTITION if and only if $f(x) \in$ BIN. Here is the function.

$a_1, \ldots, a_n$: non-negative integers.
$f(\langle a_1, a_2, \ldots, a_n \rangle)$:
1 Let $s_i = a_i$ for all i.
2 Let $T = \sum_{i \in \{1,\ldots,n\}} s_i$.
3 Let $C = T/2$.
4 Let $k = 2$.
5 Output $\langle s_1, \ldots, s_n, C, k \rangle$.

Polynomial time: Addition and division are polynomial time operations and we only do these operations polynomially many times. So the whole function is polynomial time computable.

Correctness: If x is in PARTITION, then $x = \langle a_1, a_2, \ldots, a_n \rangle$ for non-negative integers a_i, and there is $S \subset \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$. Let $T = \sum_{i \in \{1,\ldots,n\}} a_i$ be the total sum. Then we must have

$$\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j = T/2.$$

If we let $B_1 = S$ and $B_2 = \{1, \ldots, n\} \setminus S$, we see that the elements can be partitioned into two bins of capacity $T/2$. Therefore the output $f(x)$ is indeed an element of BIN.

For the other direction, if the output of the reduction $f(x)$ is an element of BIN, then there is a way to partition the elements $s_1 = a_1, \ldots, s_n = a_n$ into $k = 2$ bins B_1 and B_2 ($B_1 \cup B_2 = \{1, \ldots, n\}$, $B_1 \cap B_2 = \emptyset$) of capacity $C = T/2 = \frac{1}{2} \sum_{i \in \{1,\ldots,n\}} s_i$. Since the total sum of the elements is T and each element must be in one of the two bins, the two bins must be completely full, i.e., $\sum_{i \in B_1} a_i = T/2$ and $\sum_{j \in B_2} a_j = T/2$. If we let $S = B_1$, then $\{1, \ldots, n\} \setminus S = B_2$ and so $\sum_{i \in S} a_i = \sum_{j \in \{1,\ldots,n\} \setminus S} a_j$. So $x = \langle a_1, a_2, \ldots, a_n \rangle$ is indeed in PARTITION. ■

Chapter 13

Computational Social Choice

Exercise 13.1 (Borda count is not majority consistent)**.**
Show that Borda count voting rule is not majority consistent.

Solution. Consider the following preference profile, where there are 5 voters $\{1, 2, 3, 4, 5\}$, 3 alternatives $\{a, b, c\}$, and column i corresponds to the ranking of voter i.

1	2	3	4	5
a	a	a	b	b
b	b	b	c	c
c	c	c	a	a

In this example, alternative a is ranked first by a majority of the voters, so if the election is majority consistent, alternative a should win. However, in the Borda count voting rule, alternative b would receive 7 points (and win) whereas alternative a would receive 6 points. ■

Exercise 13.2 (Is plurality majority consistent?)**.**
Determine whether plurality is majority consistent.

Solution. Yes, plurality is majority consistent. Recall that in plurality voting rule, each voter assigns one point to the alternative that they rank first. If there is an alternative that is ranked first by a majority of the voters, then that alternative would receive more than $n/2$ votes, and no other alternative can receive more points than $n/2$ (since the total number of points awarded to all alternatives is n). ■

Exercise 13.3 (Condorcet consistency of plurality and Borda count)**.**
Determine whether plurality and Borda count voting rules are Condorcet consistent or not.

Solution. Plurality voting rule is not Condorcet consistent. To see this, consider the following preference profile, where there are 5 voters $\{1, 2, 3, 4, 5\}$, 4 alternatives $\{a, b, c, d\}$, and column i corresponds to the ranking of voter i.

1	2	3	4	5
a	a	b	c	d
c	d	a	b	b
b	b	c	a	a
d	c	d	d	c

Note that using the plurality voting rule, the winner is a. On the other hand, b is a Condorcet winner.

The above example also shows that Borda count voting rule is not Condorcet consistent as the winner would still be a. ■

Exercise 13.4 (Majority consistency vs Condorcet consistency)**.**
Does majority consistency imply Condorcet consistency? Does Condorcet consistency imply majority consistency?

Solution. Majority consistency does not imply Condorcet consistency because Plurality voting rule is majority consistent but not Condorcet consistent.

Condorcet consistency implies majority consistency. To show the claim, we need to argue that if a voting rule is Condorcet consistent, and a majority of the voters rank an alternative x first, then x must be the winner of the election. If a majority of the voters rank x first, then x is a Condorcet winner. And since the voting rule is Condorcet consistent, indeed x would be the winner of the election. ■

Exercise 13.5 (Are constant and dictatorial voting rules strategy-proof?)**.** Determine whether constant and dictatorial voting rules are strategy-proof.

Solution. Both of them are strategy-proof. If a voting rule is constant, then the winner is always fixed, so there is no way to manipulate the voting rule. If a voting rule is dictatorial, then the winner is always the dictator's first ranked alternative. So a non-dictator voter's actions do not matter, and they cannot manipulate the voting rule. Furthermore, trivially a dictator cannot manipulate the voting rule either because their first ranked alternative is always the winner. ■

Chapter 14

Approximation Algorithms

Exercise 14.1 (Optimality of the analysis of Gavril's Algorithm)**.**
Describe an infinite family of graphs for which the above algorithm returns a vertex cover which has twice the size of a minimum vertex cover.

Solution. For any $n \geq 1$, consider a perfect matching with $2n$ vertices (i.e. a set of n disjoint edges). Then the algorithm would output all the $2n$ vertices as the vertex cover. However there is clearly a vertex cover of size n (for each edge, pick one of its endpoints). This argument shows that our analysis in the proof of Theorem (Gavril's Algorithm) is tight. The algorithm is not better than a 2-approximation algorithm. In fact, note that just taking G to be a single edge allows us to conclude that the algorithm cannot be better than a 2-approximation algorithm (why?). We did not need to specify an infinite family of graphs.

Answer to 'why?': if the algorithm was a $(2-\epsilon)$-approximation algorithm for some $\epsilon > 0$, then **for all inputs**, the output of the algorithm would have to be within $(2-\epsilon)$ of the optimum. Therefore a single example where the gap is exactly factor 2 is enough to establish that the algorithm is not a $(2-\epsilon)$-approximation algorithm. ■

Exercise 14.2 (Approximation algorithm for MAX-COVERAGE)**.**
In this exercise, you will prove that there is a polynomial-time $\left(1-\frac{1}{e}\right)$-approximation algorithm for the MAX-COVERAGE problem. The algorithm you should consider is the following greedy algorithm:

```
S_1, S_2, ..., S_m: sets. k: integer in {0, 1, ..., m}.
A(⟨S_1, ..., S_m, k⟩):
1 T = ∅.
2 U = ∅. (keeping track of elements covered)
3 Repeat k times:
4    Pick j such that j ∉ T and |S_j − U| is maximized.
5    Add j to T.
6    Update U to U ∪ S_j.
7 Output T.
```

(a) Show that the algorithm runs in polynomial time.

(b) Let T^* denote the optimum solution, and let $U^* = \cup_{j \in T^*} S_j$. Note that the value of the optimum solution is $|U^*|$. Define U_i to be the set U in the above algorithm after i iterations of the loop. Let $r_i = |U^*| - |U_i|$. Prove that $r_i \leq \left(1-\frac{1}{k}\right)^i |U^*|$.

(c) Using the inequality[1] $1 - 1/k \leq e^{-1/k}$, conclude that the algorithm is a $\left(1-\frac{1}{e}\right)$-approximation algorithm for the MAX-COVERAGE problem.

Solution. Part (a): It is clear that the algorithm runs in polynomial time because each step can be carried out in polynomial time and the number of iterations of the loop is at most a polynomial.

Part (b): We prove the claim by induction on the number of iterations. In the base case, $i = 0$, and clearly

$$r_0 = |U^*| - |U_0| = |U^*| - |\emptyset| = |U^*| = \left(1-\frac{1}{k}\right)^0 |U^*|.$$

[1] This can be derived from the Taylor expansion of e^x.

For the inductive step, suppose the algorithm is entering its i'th iteration. Let t_{i-1} be the number of uncovered elements in U^* at this point. Note that $t_{i-1} \geq r_{i-1}$ (equality is achieved when all the covered elements are from U^*). We know that there are k sets that do cover all of U^*. This implies that among those k sets, at least one of them should cover at least $1/k$ fraction of the t_{i-1} uncovered elements of U^* (in other words, one of the k sets should cover at least t_{i-1}/k elements of U^*). At any given iteration, the algorithm that we provide picks the set that covers the most uncovered elements at that point. So it will pick a set that covers at least $t_{i-1}/k \geq r_{i-1}/k$ elements. Therefore

$$r_i \leq r_{i-1} - \frac{r_{i-1}}{k} = \left(1 - \frac{1}{k}\right) r_{i-1}.$$

Combining the induction hypothesis $r_{i-1} \leq (1 - 1/k)^{i-1}|U^*|$ with the above inequality, we can bound r_i as follows:

$$r_i \leq \left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{k}\right)^{i-1} |U^*| = (1 - k)^i |U^*|.$$

Part (c): When the algorithm terminates, we have (using the given inequality)

$$r_k \leq \left(1 - \frac{1}{k}\right)^k |U^*| \leq e^{-1}|U^*|.$$

Recall $r_k = |U^*| - |U_k|$, where $|U_k|$ is the value of the solution that the algorithm outputs (i.e. the number of elements covered by the algorithm). The optimum value is $|U^*|$. Combining everything, we get a lower bound on the number of elements covered by the algorithm in terms of the optimum value $|U^*|$:

$$|U_k| \geq (1 - e^{-1})|U^*|.$$

This establishes that the algorithm is a $(1 - e^{-1})$-approximation algorithm. ■

Exercise 14.3 (Approximation algorithm for MIN-SET-COVER)**.**
In the *set-cover problem*, the input is a set X together with a collection of (possibly intersecting) subsets $S_1, S_2, \ldots, S_m \subseteq X$ (we assume the union of all the sets is X). The output is a minimum size set $T \subseteq \{1, 2, \ldots, m\}$ such that $\cup_{i \in T} S_i = X$. We denote this problem by MIN-SET-COVER. Give a polynomial-time $(\ln |X|)$-approximation algorithm for this problem.

Solution. We sketch the argument.

The algorithm is essentially the same as the one given in the previous exercise. The only difference is that we don't repeat for a fixed number of times but we repeat until we cover every element.

Let k^* be the number of sets in an optimum solution. Using a similar argument to the one given in the previous exercise, we can show by induction that after i iterations of the algorithm, the number of uncovered elements is at most $|X|(1 - 1/k^*)^i$. So after $k^* \ln |X|$ iterations, the number of uncovered elements must be strictly less than 1 since

$$|X| \left(1 - \frac{1}{k^*}\right)^{k^* \ln |X|} < |X| e^{-\ln |X|} = 1.$$

So the algorithm terminates with an output of at most $k^* \ln |X|$ sets, which is within a factor of $\ln |X|$ of the optimum. ■

Chapter 15

Probability Theory

Exercise 15.1 (Probability space modeling)**.**
How would you model a roll of a single 6-sided die using Definition (Finite probability space, sample space, probability distribution)? How about a roll of two dice? How about a roll of a die and a coin toss together?

Solution. For the case of a single 6-sided die, we want the model to match our intuitive understanding and real-world experience that the probability of observing each of the possible die rolls $1, 2, \ldots, 6$ is equal. We formalize this by defining the *sample space* Ω and the *probability distribution* $\mathbf{Pr}$ as follows:

$$\Omega = \{1, 2, 3, 4, 5, 6\}, \qquad \mathbf{Pr}[\ell] = \frac{1}{6} \text{ for all } \ell \in \Omega.$$

Similarly, to model a roll of two dice, we can let each outcome be an ordered pair representing the roll of each of the two dice:

$$\Omega = \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\}, \qquad \mathbf{Pr}[\ell] = \left(\frac{1}{6}\right)^2 = \frac{1}{36} \text{ for all } \ell \in \Omega.$$

Lastly, to model a roll of a die and a coin toss together, we can let each outcome be an ordered pair where the first element represents the result of the die roll, and the second element represents the result of the coin toss:

$$\Omega = \{1, 2, 3, 4, 5, 6\} \times \{\text{Heads}, \text{Tails}\}, \qquad \mathbf{Pr}[\ell] = \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{12} \text{ for all } \ell \in \Omega.$$

■

Exercise 15.2 (Practice with events)**.**

(a) Suppose we roll two 6-sided dice. How many different events are there? Write down the event corresponding to "we roll a double" and determine its probability.

(b) Suppose we roll a 3-sided die and see the number d. We then roll a d-sided die. How many different events are there? Write down the event corresponding to "the second roll is a 2" and determine its probability.

Solution. Part (a): We use the model for rolling two 6-sided dice as in Exercise (Probability space modeling). Since an event is any subset of outcomes $E \subseteq \Omega$, the number of events is the number of such subsets, which is $|\mathcal{P}(\Omega)| = 2^{|\Omega|} = 2^{36}$ (here $\mathcal{P}(\Omega)$ denotes the power set of Ω).

The event corresponding to "we roll a double" can be expressed as

$$E = \big\{(\ell, \ell) : \ell \in \{1, 2, 3, 4, 5, 6\}\big\}$$

which has probability

$$\mathbf{Pr}[E] = \sum_{\ell \in E} \mathbf{Pr}[\ell] = \frac{1}{36} \cdot |E| = \frac{6}{36} = \frac{1}{6}.$$

Part(b): We model the two dice rolls as follows:

$$\Omega = \big\{(a, b) \in \{1, 2, 3\}^2 : a \geq b\big\}, \qquad \mathbf{Pr}[(a, b)] = \frac{1}{3} \cdot \frac{1}{a} = \frac{1}{3a}.$$

The restriction that $a \geq b$ is imposed because the second die depends on the first roll and the result of the second roll cannot be larger than that of the first. Note that we could also have used a model where $\Omega = \{1, 2, 3\}^2$ and assigned a probability of 0 to the outcomes where $a < b$, but considering outcomes that never occur is typically not very useful.

In the model we originally defined, the number of events is $|\mathcal{P}(\Omega)| = 2^{|\Omega|} = 2^6 = 64$. Note that the number of events depends on the size of the sample space, so this number can vary depending on the model.

The event corresponding to "the second roll is a 2" is given by

$$E = \{(a, b) \in \Omega : b = 2\} = \{(2, 2), (3, 2)\}$$

which has probability

$$\mathbf{Pr}[E] = \sum_{\ell \in E} \mathbf{Pr}[\ell] = \mathbf{Pr}[(2, 2)] + \mathbf{Pr}[(3, 2)] = \frac{1}{3 \cdot 2} + \frac{1}{3 \cdot 3} = \frac{5}{18}.$$

■

Exercise 15.3 (Basic facts about probability)**.**
Let A and B be two events. Prove the following.

- If $A \subseteq B$, then $\mathbf{Pr}[A] \leq \mathbf{Pr}[B]$.
- $\mathbf{Pr}[\overline{A}] = 1 - \mathbf{Pr}[A]$.
- $\mathbf{Pr}[A \cup B] = \mathbf{Pr}[A] + \mathbf{Pr}[B] - \mathbf{Pr}[A \cap B]$.

Solution. Part (a): Suppose $A \subseteq B$. Recall that $\mathbf{Pr}$ is a nonnegative function (i.e. $\mathbf{Pr}[\ell] \geq 0$ for all $\ell \in \Omega$). Hence,

$$\begin{aligned}
\mathbf{Pr}[A] &= \sum_{\ell \in A} \mathbf{Pr}[\ell] && \text{(by Definition (Event))} \\
&\leq \sum_{\ell \in A} \mathbf{Pr}[\ell] + \sum_{\ell \in B \setminus A} \mathbf{Pr}[\ell] && \text{(by nonnegativity of } \mathbf{Pr}) \\
&= \sum_{\ell \in B} \mathbf{Pr}[\ell] \\
&= \mathbf{Pr}[B].
\end{aligned}$$

Part (b): Recall that $\overline{A}$ denotes $\Omega \setminus A$, and that $\sum_{\ell \in \Omega} \mathbf{Pr}[\ell] = 1$. Hence,

$$\begin{aligned}
\mathbf{Pr}[\overline{A}] &= \sum_{\ell \in \overline{A}} \mathbf{Pr}[\ell] && \text{(by Definition (Event))} \\
&= \sum_{\ell \in \Omega \setminus A} \mathbf{Pr}[\ell] \\
&= \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] - \sum_{\ell \in A} \mathbf{Pr}[\ell] \\
&= 1 - \mathbf{Pr}[A].
\end{aligned}$$

(by Definition (Finite probability space, sample space, probability distribution))

Part (c) By partitioning $A \cup B$ into $A \setminus B$, $B \setminus A$, and $A \cap B$, we see that

$$\begin{aligned}
\mathbf{Pr}[A \cup B] &= \sum_{\ell \in A \cup B} \mathbf{Pr}[\ell] && \text{(by Definition (Event))} \\
&= \sum_{\ell \in A \setminus B} \mathbf{Pr}[\ell] + \sum_{\ell \in B \setminus A} \mathbf{Pr}[\ell] + \sum_{\ell \in A \cap B} \mathbf{Pr}[\ell] \\
&= \left(\sum_{\ell \in A} \mathbf{Pr}[\ell] - \sum_{\ell \in A \cap B} \mathbf{Pr}[\ell] \right) + \left(\sum_{\ell \in B} \mathbf{Pr}[\ell] - \sum_{\ell \in A \cap B} \mathbf{Pr}[\ell] \right) + \sum_{\ell \in A \cap B} \mathbf{Pr}[\ell] \\
&= \sum_{\ell \in A} \mathbf{Pr}[\ell] + \sum_{\ell \in B} \mathbf{Pr}[\ell] - \sum_{\ell \in A \cap B} \mathbf{Pr}[\ell] \\
&= \mathbf{Pr}[A] + \mathbf{Pr}[B] - \mathbf{Pr}[A \cap B]. && \text{(by Definition (Event))}
\end{aligned}$$

■

Exercise 15.4 (Union bound)**.**
Let $A_1, A_2, \ldots, A_n$ be events. Then

$$\mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_n] \leq \mathbf{Pr}[A_1] + \mathbf{Pr}[A_2] + \cdots + \mathbf{Pr}[A_n].$$

We get equality if and only if the A_i's are pairwise disjoint.

Solution. By the last part of Exercise (Basic facts about probability), we can conclude that

$$\mathbf{Pr}[A \cup B] \leq \mathbf{Pr}[A] + \mathbf{Pr}[B].$$

We also notice that equality holds if and only if $\mathbf{Pr}[A \cap B] = 0$, which happens if and only if A and B are disjoint. We will extend this by induction on n.

The expression is identical on both sides for the case where $n = 1$, and the case where $n = 2$ is exactly as above. These serve as the base cases for our induction.

For the inductive case, assume the proposition holds true for $n = k$. We seek to show that it too holds true for $n = k+1$. Given events $A_1, A_2, \ldots, A_k, A_{k+1}$, let

$$A = A_1 \cup A_2 \cup \cdots \cup A_k, \qquad B = A_{k+1}.$$

Then

$$\begin{aligned}
\mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_k \cup A_{k+1}] &= \mathbf{Pr}[A \cup B] \\
&\leq \mathbf{Pr}[A] + \mathbf{Pr}[B] \qquad \text{(by the above result)} \\
&= \mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_k] + \mathbf{Pr}[A_{k+1}] \\
&\leq (\mathbf{Pr}[A_1] + \cdots + \mathbf{Pr}[A_k]) + \mathbf{Pr}[A_{k+1}] \\
&\qquad \text{(by the induction hypothesis)}
\end{aligned}$$

where equality holds if and only if A and B are disjoint and $A_1, \ldots, A_k$ are pairwise disjoint. In other words, equality holds if and only if

$$\bigcup_{t=1}^{k} (A_t \cap A_{k+1}) = \emptyset \qquad \text{and} \qquad A_i \cap A_j = \emptyset \text{ for all } 1 \leq i < j \leq k,$$

which in turn holds if and only if

$$A_i \cap A_j = \emptyset \text{ for all } 1 \leq i < j \leq k+1.$$

(i.e. $A_1, \ldots, A_{k+1}$ are pairwise disjoint). This completes the proof. ■

Exercise 15.5 (Conditional probability practice)**.**
Suppose we roll a 3-sided die and see the number d. We then roll a d-sided die. We are interested in the probability that the first roll was a 1 given that the second roll was a 1. First express this probability using the notation of conditional probability and then determine what the probability is.

Solution. We use the model defined in Exercise (Practice with events). The event that the first roll is a 1 is

$$E_1 = \{(1,1)\}$$

and the event that the second roll is a 1 is

$$E_2 = \{(1,1), (2,1), (3,1)\}$$

with corresponding probabilities

$$\mathbf{Pr}[E_1] = \frac{1}{3}, \qquad \mathbf{Pr}[E_2] = \frac{1}{3 \cdot 1} + \frac{1}{3 \cdot 2} + \frac{1}{3 \cdot 3} = \frac{11}{18}.$$

Then the conditional probability we are interested in is

$$\mathbf{Pr}[E_1 \mid E_2] = \frac{\mathbf{Pr}[E_1 \cap E_2]}{\mathbf{Pr}[E_2]} = \frac{\mathbf{Pr}[E_1]}{\mathbf{Pr}[E_2]} = \frac{1/3}{11/18} = \frac{6}{11}.$$

■

Exercise 15.6 (Practice with chain rule)**.**
Suppose there are 100 students in 15-251 and 5 of the students are Trump supporters. We pick 3 students from class uniformly at random. Calculate the probability that none of them are Trump supporters using Proposition (Chain rule).

Solution. For $i = 1, 2, 3$, let A_i be the event that the i-th student we pick is not a Trump supporter. Then using the chain rule, the probability that none of them are Trump supporters is

$$\mathbf{Pr}[A_1 \cap A_2 \cap A_3] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[A_2 \mid A_1] \cdot \mathbf{Pr}[A_3 \mid A_1 \cap A_2] = \frac{95}{100} \cdot \frac{94}{99} \cdot \frac{93}{98}.$$

■

Exercise 15.7 (Proof of the law of total probability)**.**
Prove the above proposition.

Solution. We note that

$$B = B \cap \Omega = B \cap (A_1 \cup A_2 \cup \cdots \cup A_n) = (B \cap A_1) \cup (B \cap A_2) \cup \cdots \cup (B \cap A_n),$$

with the $(B \cap A_i)$'s being pairwise disjoint. The first equality

$$\mathbf{Pr}[B] = \mathbf{Pr}[B \cap A_1] + \mathbf{Pr}[B \cap A_2] + \cdots + \mathbf{Pr}[B \cap A_n]$$

follows from a direct application of Exercise **??**. The equivalent statement

$$\mathbf{Pr}[B] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[B \mid A_1] + \mathbf{Pr}[A_2] \cdot \mathbf{Pr}[B \mid A_2] + \cdots + \mathbf{Pr}[A_n] \cdot \mathbf{Pr}[B \mid A_n]$$

follows from Definition (Conditional probability). ■

Exercise 15.8 (Practice with the law of total probability)**.**
There are 2 bins. Bin 1 contains 6 red balls and 4 blue balls. Bin 2 contains 3 red balls and 7 blue balls. We choose a bin uniformly at random, and then choose one of the balls in that bin uniformly at random. Calculate the probability that the chosen ball is red using Proposition (Law of total probability).

Solution. Let A_1 and A_2 be the events that we pick the first and second bin respectively. Note that A_1 and A_2 partition the sample space, and $\mathbf{Pr}[A_1] = \mathbf{Pr}[A_2] = \frac{1}{2}$. Let B be the event that the chosen ball is red. Then using the law of total probability,

$$\mathbf{Pr}[B] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[B \mid A_1] + \mathbf{Pr}[A_2] \cdot \mathbf{Pr}[B \mid A_2] = \frac{1}{2} \cdot \frac{6}{10} + \frac{1}{2} \cdot \frac{3}{10} = \frac{9}{20}.$$

■

Exercise 15.9 (Practice with Bayes' rule)**.**
CAPTCHAs are tests designed to be hard for computers to solve but easy for people to solve. Suppose it is estimated that $3/4$ of all attempts to solve a CAPTCHA are from humans and the remainder are from computers. If a human has a $9/10$ chance of successfully solving a CAPTCHA and a computer has a $1/5$ chance, what is the probability that the entity attempting a CAPTCHA was a human, given that the CAPTCHA was successfully solved?

Solution. Let A be the event that the entity attempting a CAPTCHA was a human, and let B be the event that the CAPTCHA was successfully solved. Then we are interested in the probability

$$\begin{aligned}
\mathbf{Pr}[A \mid B] &= \frac{\mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A]}{\mathbf{Pr}[B]} && \text{(by Bayes' rule)} \\
&= \frac{\mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A]}{\mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A] + \mathbf{Pr}[\overline{A}] \cdot \mathbf{Pr}[B \mid \overline{A}]} \\
& && \text{(by the law of total probability)} \\
&= \frac{\frac{3}{4} \cdot \frac{9}{10}}{\frac{3}{4} \cdot \frac{9}{10} + \frac{1}{4} \cdot \frac{1}{5}} \\
&= \frac{27}{29}.
\end{aligned}$$

■

Exercise 15.10 (Pair-wise independent but not three-wise)**.**
Give an example of a probability space with 3 events A_1, A_2 and A_3 such that each pair of events A_i and A_j are independent, however A_1, A_2, A_3 together are dependent.

Solution. Consider the following probability space:

$$\Omega = \{1, 2, 3, 4\}, \qquad \mathbf{Pr}[\ell] = \frac{1}{4} \text{ for all } \ell \in \Omega.$$

Define the events $A_1 = \{2, 3\}$, $A_2 = \{1, 3\}$, and $A_3 = \{1, 2\}$. Note that

$$\mathbf{Pr}[A_1] = \mathbf{Pr}[A_2] = \mathbf{Pr}[A_3] = \frac{1}{2}$$

and that for any $i, j \in \{1, 2, 3\}$ with $i \neq j$, we have $|A_i \cap A_j| = 1$, and so

$$\mathbf{Pr}[A_i \cap A_j] = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = \mathbf{Pr}[A_i] \cdot \mathbf{Pr}[A_j].$$

This shows that A_1, A_2, A_3 are pairwise independent. However,

$$\mathbf{Pr}[A_1 \cap A_2 \cap A_3] = \mathbf{Pr}[\emptyset] = 0 \neq \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[A_2] \cdot \mathbf{Pr}[A_3],$$

so A_1, A_2, A_3 are dependent. ■

Exercise 15.11 (Practice with random variables)**.**
Suppose we roll two 6-sided dice. Let $\boldsymbol{X}$ be the random variable that denotes the sum of the numbers we see. Explicitly write down the input-output pairs for the function $\boldsymbol{X}$. Calculate $\mathbf{Pr}[\boldsymbol{X} \geq 7]$.

Solution. We use the model for rolling two 6-sided dice as in Exercise (Probability space modeling). Then

$$\begin{array}{llllll}
\boldsymbol{X}(1,1)=2, & \boldsymbol{X}(1,2)=3, & \boldsymbol{X}(1,3)=4, & \boldsymbol{X}(1,4)=5, & \boldsymbol{X}(1,5)=6, & \boldsymbol{X}(1,6)=7,\\
\boldsymbol{X}(2,1)=3, & \boldsymbol{X}(2,2)=4, & \boldsymbol{X}(2,3)=5, & \boldsymbol{X}(2,4)=6, & \boldsymbol{X}(2,5)=7, & \boldsymbol{X}(2,6)=8,\\
\boldsymbol{X}(3,1)=4, & \boldsymbol{X}(3,2)=5, & \boldsymbol{X}(3,3)=6, & \boldsymbol{X}(3,4)=7, & \boldsymbol{X}(3,5)=8, & \boldsymbol{X}(3,6)=9,\\
\boldsymbol{X}(4,1)=5, & \boldsymbol{X}(4,2)=6, & \boldsymbol{X}(4,3)=7, & \boldsymbol{X}(4,4)=8, & \boldsymbol{X}(4,5)=9, & \boldsymbol{X}(4,6)=10,\\
\boldsymbol{X}(5,1)=6, & \boldsymbol{X}(5,2)=7, & \boldsymbol{X}(5,3)=8, & \boldsymbol{X}(5,4)=9, & \boldsymbol{X}(5,5)=10, & \boldsymbol{X}(5,6)=11,\\
\boldsymbol{X}(6,1)=7, & \boldsymbol{X}(6,2)=8, & \boldsymbol{X}(6,3)=9, & \boldsymbol{X}(6,4)=10, & \boldsymbol{X}(6,5)=11, & \boldsymbol{X}(6,6)=12.
\end{array}$$

Since the probability distribution is uniform over the outcomes,

$$\mathbf{Pr}[\boldsymbol{X}\geq 7]=\frac{1}{36}\cdot|\{\ell\in\Omega:\boldsymbol{X}(\ell)\geq 7\}|=\frac{1}{36}\cdot 21=\frac{7}{12}.$$

■

Exercise 15.12 (Facts about probability mass function)**.**
Verify the following:

- $\sum_{x\in\text{range}(\boldsymbol{X})} p_{\boldsymbol{X}}(x)=1$,
- for $S\subseteq\mathbb{R}$, $\mathbf{Pr}[\boldsymbol{X}\in S]=\sum_{x\in S} p_{\boldsymbol{X}}(x)$.

Solution. Part (a): $\sum_{x\in\text{range}(\boldsymbol{X})} p_{\boldsymbol{X}}(x)=1$

$$\begin{aligned}
\sum_{x\in\text{range}(\boldsymbol{X})} p_{\boldsymbol{X}}(x) &= \sum_{x\in\text{range}(\boldsymbol{X})} \mathbf{Pr}[\boldsymbol{X}=x]\\
&\qquad\text{(by Definition (Probability mass function))}\\
&= \sum_{x\in\text{range}(\boldsymbol{X})} \mathbf{Pr}[\{\ell\in\Omega:\boldsymbol{X}(\ell)=x\}]\\
&\text{(by Definition (Common events through a random variable))}\\
&= \sum_{x\in\text{range}(\boldsymbol{X})}\ \sum_{\substack{\ell\in\Omega\\ \boldsymbol{X}(\ell)=x}} \mathbf{Pr}[\ell] \qquad \text{(by Definition (Event))}\\
&= \sum_{\ell\in\Omega}\mathbf{Pr}[\ell]\\
&\text{(since } \big\{\{\ell\in\Omega:\boldsymbol{X}(\ell)=x\}:x\in\text{range}(\boldsymbol{X})\big\} \text{ partitions } \Omega)\\
&= 1.
\end{aligned}$$

Part (b): $\mathbf{Pr}[\boldsymbol{X}\in S]=\sum_{x\in S} p_{\boldsymbol{X}}(x)$

$$\begin{aligned}
\mathbf{Pr}[\boldsymbol{X}\in S] &= \mathbf{Pr}[\{\ell\in\Omega:\boldsymbol{X}(\ell)\in S\}]\\
&\text{(by Definition (Common events through a random variable))}\\
&= \sum_{\substack{\ell\in\Omega\\ \boldsymbol{X}(\ell)\in S}} \mathbf{Pr}[\ell] \qquad \text{(by Definition (Event))}\\
&= \sum_{x\in S}\ \sum_{\substack{\ell\in\Omega\\ \boldsymbol{X}(\ell)=x}} \mathbf{Pr}[\ell] \qquad \text{(by splitting the summation over } x\in S)\\
&= \sum_{x\in S}\mathbf{Pr}[\{\ell\in\Omega:\boldsymbol{X}(\ell)=x\}] \qquad \text{(by Definition (Event))}\\
&= \sum_{x\in S}\mathbf{Pr}[\boldsymbol{X}=x]\\
&\text{(by Definition (Common events through a random variable))}\\
&= \sum_{x\in S} p_{\boldsymbol{X}}(x). \qquad \text{(by Definition (Probability mass function))}
\end{aligned}$$

■

Exercise 15.13 (Equivalence of expected value definitions)**.**
Prove that the above two expressions for $\mathbf{E}[\boldsymbol{X}]$ are equivalent.

Solution. We show a chain of equalities from the RHS to the LHS:

$$\sum_{x \in \text{range}(\boldsymbol{X})} \mathbf{Pr}[\boldsymbol{X} = x] \cdot x = \sum_{x \in \text{range}(\boldsymbol{X})} \mathbf{Pr}[\{\ell \in \Omega : \boldsymbol{X}(\ell) = x\}] \cdot x$$

(by Definition (Common events through a random variable))

$$= \sum_{x \in \text{range}(\boldsymbol{X})} \sum_{\substack{\ell \in \Omega \\ \boldsymbol{X}(\ell) = x}} \mathbf{Pr}[\ell] \cdot x$$

(by Definition (Probability mass function))

$$= \sum_{x \in \text{range}(\boldsymbol{X})} \sum_{\substack{\ell \in \Omega \\ \boldsymbol{X}(\ell) = x}} \mathbf{Pr}[\ell] \cdot \boldsymbol{X}(\ell)$$

(since $\boldsymbol{X}(\ell) = x$ in the second summation)

$$= \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot \boldsymbol{X}(\ell).$$

■

Exercise 15.14 (Practice with expected value)**.**
Suppose we roll two 6-sided dice. Let $\boldsymbol{X}$ be the random variable that denotes the sum of the numbers we see. Calculate $\mathbf{E}[\boldsymbol{X}]$.

Solution. We refer to the input-output pairs of $\boldsymbol{X}$ (recall that random variables are functions) in Exercise (Practice with random variables). With a lot of tedious calculations, we can compute

$$\mathbf{E}[\boldsymbol{X}] = \sum_{x \in \text{range}(\boldsymbol{X})} \mathbf{Pr}[\boldsymbol{X} = x] \cdot x = \frac{1}{36} \cdot 2 + \frac{2}{36} \cdot 3 + \frac{3}{36} \cdot 4 + \cdots + \frac{1}{36} \cdot 12 = 7.$$

We will see a less tedious way of performing such calculations in the following exercise. ■

Exercise 15.15 (Practice with linearity of expectation)**.**
Suppose we roll three 10-sided dice. Let $\boldsymbol{X}$ be the sum of the three values we see. Calculate $\mathbf{E}[\boldsymbol{X}]$.

Solution. Let $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$ be the values of the rolls of each of the three dice. Note that $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$ are random variables and that $\boldsymbol{X} = \boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3$. Then since $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$ are identically distributed, we can compute

$$\mathbf{E}[\boldsymbol{X}_1] = \mathbf{E}[\boldsymbol{X}_2] = \mathbf{E}[\boldsymbol{X}_3] = \sum_{x \in \text{range}(\boldsymbol{X}_1)} \mathbf{Pr}[\boldsymbol{X}_1 = x] \cdot x \qquad \text{(by Definition ??)}$$

$$= \sum_{x=1}^{10} \mathbf{Pr}[\boldsymbol{X}_1 = x] \cdot x$$

(since $\text{range}(\boldsymbol{X}_1) = \{1, 2, \ldots, 10\}$)

$$= \sum_{x=1}^{10} \frac{1}{10} \cdot x \qquad \text{(since the die is fair)}$$

$$= \frac{11}{2}$$

and by linearity of expectation,

$$\mathbf{E}[\boldsymbol{X}] = \mathbf{E}[\boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3] = \mathbf{E}[\boldsymbol{X}_1] + \mathbf{E}[\boldsymbol{X}_2] + \mathbf{E}[\boldsymbol{X}_3] = 3 \cdot \frac{11}{2} = \frac{33}{2}.$$

■

Exercise 15.16 (Expectation of product of random variables)**.**
Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be random variables. Is it always true that $\mathbf{E}[\boldsymbol{XY}] = \mathbf{E}[\boldsymbol{X}]\,\mathbf{E}[\boldsymbol{Y}]$?

Solution. No. Consider the experiment of flipping a single fair coin, and the corresponding model where $\Omega = \{\text{Heads}, \text{Tails}\}$ and $\mathbf{Pr}[\text{Heads}] = \mathbf{Pr}[\text{Tails}] = 1/2$. Define the random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ such that

$$\boldsymbol{X}(\ell) = \begin{cases} 1 & \text{if } \ell = \text{Heads}, \\ 0 & \text{if } \ell = \text{Tails}, \end{cases} \qquad \boldsymbol{Y}(\ell) = \begin{cases} 1 & \text{if } \ell = \text{Tails}, \\ 0 & \text{if } \ell = \text{Heads}. \end{cases}$$

One can verify that $\mathbf{E}[\boldsymbol{X}] = \mathbf{E}[\boldsymbol{Y}] = 1/2$, but $\boldsymbol{XY} \equiv 0$ since regardless of whether the outcome is Heads or Tails, one of $\boldsymbol{X}$ and $\boldsymbol{Y}$ evaluates to 0, so their product is always 0. So

$$\mathbf{E}[\boldsymbol{XY}] = 0 \neq \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = \mathbf{E}[\boldsymbol{X}]\,\mathbf{E}[\boldsymbol{Y}].$$

■

Exercise 15.17 (Practice with linearity of expectation and indicators)**.**

(a) There are n balls and n bins. For each ball, you pick one of the bins uniformly at random and drop the ball in that bin. What is the expected number of balls in bin 1? What is the expected number of empty bins?

(b) Suppose you randomly color the vertices of the complete graph on n vertices one of k colors. What is the expected number of paths of length c (where we assume $c \geq 3$) such that no two adjacent vertices on the path have the same color?

Solution. Part (a): Let $\boldsymbol{X}$ be the number of balls in bin 1. For $j = 1, 2, \ldots, n$, let E_j be the event that the j-th ball is dropped in bin 1. Observe that $\boldsymbol{X} = \sum_{j=1}^{n} \boldsymbol{I}_{E_j}$, and that $\mathbf{Pr}[E_j] = 1/n$ for all j, since the bin each ball is dropped into is picked uniformly at random. Then by linearity of expectation,

$$\mathbf{E}[\boldsymbol{X}] = \mathbf{E}\left[\sum_{j=1}^{n} \boldsymbol{I}_{E_j}\right] = \sum_{j=1}^{n} \mathbf{E}[\boldsymbol{I}_{E_j}] = \sum_{j=1}^{n} \mathbf{Pr}[E_j] = \sum_{j=1}^{n} \frac{1}{n} = 1.$$

Let $\boldsymbol{Y}$ be the number of empty bins. For $j = 1, 2, \ldots, n$, let F_j be the event that bin j is empty. Observe that $\boldsymbol{Y} = \sum_{j=1}^{n} \boldsymbol{I}_{F_j}$, and that $\mathbf{Pr}[F_j] = (1 - 1/n)^n$ for all j, since the probability that any one of the n balls is *not* dropped in a fixed bin is $1 - 1/n$, and each ball is dropped independently of the others. Then by linearity of expectation,

$$\mathbf{E}[\boldsymbol{Y}] = \mathbf{E}\left[\sum_{j=1}^{n} \boldsymbol{I}_{F_j}\right] = \sum_{j=1}^{n} \mathbf{E}[\boldsymbol{I}_{F_j}] = \sum_{j=1}^{n} \mathbf{Pr}[F_j] = \sum_{j=1}^{n} \left(1 - \frac{1}{n}\right)^n = n\left(1 - \frac{1}{n}\right)^n.$$

Part (b): Let $\boldsymbol{X}$ be the number of paths of length c such that no two adjacent vertices on the path have the same color. There are a total of

$$N = n(n-1)\cdots(n-c) = \frac{n!}{(n-c-1)!}$$

paths of length c. Number the paths from 1 to N, and for $j = 1, 2, \ldots, N$, let E_j be the event that no two adjacent vertices on the j-th path have the same color. Note that $\boldsymbol{X} = \sum_{j=1}^{N} \boldsymbol{I}_{E_j}$.

We first compute $\mathbf{Pr}[E_j]$ for some fixed j. Suppose path j is $v_0 v_1 \cdots v_c$. Then E_j occurs if and only if v_i is colored differently from v_{i-1} for $i = 1, 2, \ldots, c$. For each i, this happens with probability $1 - 1/k$, and they are independent of each other. Hence we can conclude that $\mathbf{Pr}[E_j] = (1 - 1/k)^c$ for each j. Then by linearity of expectation,

$$\mathbf{E}[\boldsymbol{X}] = \mathbf{E}\left[\sum_{j=1}^{N} \boldsymbol{I}_{E_j}\right] = \sum_{j=1}^{N} \mathbf{E}[\boldsymbol{I}_{E_j}] = \sum_{j=1}^{N} \mathbf{Pr}[E_j] = \sum_{j=1}^{N} \left(1 - \frac{1}{k}\right)^c = \frac{n!}{(n-c-1)!}\left(1 - \frac{1}{k}\right)^c.$$

■

Exercise 15.18 (Proof of law of total expectation)**.**
Prove the above proposition.

Solution. We show a chain of equalities from the RHS to the LHS:

$$\begin{aligned}
\sum_{j=1}^{n} \mathbf{E}[\boldsymbol{X} \mid A_j] \cdot \mathbf{Pr}[A_j] &= \sum_{j=1}^{n} \sum_{x \in \mathrm{range}(\boldsymbol{X})} x \cdot \mathbf{Pr}[\boldsymbol{X} = x \mid A_j] \cdot \mathbf{Pr}[A_j] \\
&\qquad \text{(by Definition (Conditional expectation))} \\
&= \sum_{j=1}^{n} \sum_{x \in \mathrm{range}(\boldsymbol{X})} x \cdot \frac{\mathbf{Pr}[(\boldsymbol{X} = x) \cap A_j]}{\mathbf{Pr}[A_j]} \cdot \mathbf{Pr}[A_j] \\
&\qquad \text{(by Definition (Conditional probability))} \\
&= \sum_{j=1}^{n} \sum_{x \in \mathrm{range}(\boldsymbol{X})} x \cdot \mathbf{Pr}[(\boldsymbol{X} = x) \cap A_j] \\
&\qquad \text{(by cancelling } \mathbf{Pr}[A_j]) \\
&= \sum_{x \in \mathrm{range}(\boldsymbol{X})} \left(x \cdot \sum_{j=1}^{n} \mathbf{Pr}[(\boldsymbol{X} = x) \cap A_j] \right) \\
&\qquad \text{(by moving the outer summation inside)} \\
&= \sum_{x \in \mathrm{range}(\boldsymbol{X})} x \cdot \mathbf{Pr}[\boldsymbol{X} = x] \\
&\qquad \text{(by Proposition (Law of total probability))} \\
&= \mathbf{E}[\boldsymbol{X}] \\
&\text{(by Definition (Expected value of a random variable))}
\end{aligned}$$

■

Exercise 15.19 (Practice with law of total expectation)**.**
We first roll a 4-sided die. If we see the value d, we then roll a d-sided die. Let $\boldsymbol{X}$ be the sum of the two values we see. Calculate $\mathbf{E}[\boldsymbol{X}]$.

Solution. Let $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ be the value of the first and second roll respectively. Note that $\boldsymbol{X} = \boldsymbol{X}_1 + \boldsymbol{X}_2$ and that

$$\mathbf{E}[\boldsymbol{X}_2 \mid \boldsymbol{X}_1 = d] = \sum_{x \in \mathrm{range}(\boldsymbol{X}_2)} x \cdot \mathbf{Pr}[\boldsymbol{X}_2 = x \mid \boldsymbol{X}_1 = d] = \sum_{x=1}^{d} x \cdot \frac{1}{d} = \frac{d+1}{2}.$$

Applying linearity of expectation to the modified probability space $(\Omega, \mathbf{Pr}_{\{\boldsymbol{X}_1=d\}})$, we get

$$\begin{aligned}\mathbf{E}[\boldsymbol{X} \mid \boldsymbol{X}_1 = d] &= \mathbf{E}[\boldsymbol{X}_1 + \boldsymbol{X}_2 \mid \boldsymbol{X}_1 = d] \\ &= \mathbf{E}[\boldsymbol{X}_1 \mid \boldsymbol{X}_1 = d] + \mathbf{E}[\boldsymbol{X}_2 \mid \boldsymbol{X}_1 = d] \\ &= d + \frac{d+1}{2} \\ &= \frac{3d+1}{2}.\end{aligned}$$

Then by the law of total expectation,

$$\mathbf{E}[\boldsymbol{X}] = \sum_{d=1}^{4} \mathbf{E}[\boldsymbol{X} \mid \boldsymbol{X}_1 = d] \cdot \mathbf{Pr}[\boldsymbol{X}_1 = d] = \sum_{d=1}^{4} \frac{3d+1}{2} \cdot \frac{1}{4} = \frac{17}{4}.$$

■

Exercise 15.20 (Expectation of product of independent random variables)**.** Show that if $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n$ are independent random variables, then

$$\mathbf{E}[\boldsymbol{X}_1 \boldsymbol{X}_2 \cdots \boldsymbol{X}_n] = \mathbf{E}[\boldsymbol{X}_1] \cdot \mathbf{E}[\boldsymbol{X}_2] \cdots \mathbf{E}[\boldsymbol{X}_n].$$

Solution. We will first show the following sub-claim: if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent random variables, then

$$\mathbf{E}[\boldsymbol{X}\boldsymbol{Y}] = \mathbf{E}[\boldsymbol{X}] \cdot \mathbf{E}[\boldsymbol{Y}].$$

Indeed, noting that the events $(\boldsymbol{X} = x) \cap (\boldsymbol{Y} = y)$, for $x \in \operatorname{range}(\boldsymbol{X})$ and $y \in \operatorname{range}(\boldsymbol{Y})$, partition Ω,

$$\begin{aligned}\mathbf{E}[\boldsymbol{X}\boldsymbol{Y}] &= \sum_{x \in \operatorname{range}(\boldsymbol{X})} \sum_{y \in \operatorname{range}(\boldsymbol{Y})} \mathbf{E}[\boldsymbol{X}\boldsymbol{Y} \mid (\boldsymbol{X} = x) \cap (\boldsymbol{Y} = y)] \cdot \mathbf{Pr}[(\boldsymbol{X} = x) \cap (\boldsymbol{Y} = y)] \\ &\qquad \text{(by Proposition (Law of total expectation))} \\ &= \sum_{x \in \operatorname{range}(\boldsymbol{X})} \sum_{y \in \operatorname{range}(\boldsymbol{Y})} xy \cdot \mathbf{Pr}[(\boldsymbol{X} = x) \cap (\boldsymbol{Y} = y)] \\ &\qquad \text{(by Definition (Conditional expectation))} \\ &= \sum_{x \in \operatorname{range}(\boldsymbol{X})} \sum_{y \in \operatorname{range}(\boldsymbol{Y})} xy \cdot \mathbf{Pr}[\boldsymbol{X} = x] \cdot \mathbf{Pr}[\boldsymbol{Y} = y] \\ &\qquad \text{(by independence of the events } \boldsymbol{X} = x \text{ and } \boldsymbol{Y} = y) \\ &= \left(\sum_{x \in \operatorname{range}(\boldsymbol{X})} x \cdot \mathbf{Pr}[\boldsymbol{X} = x] \right) \cdot \left(\sum_{y \in \operatorname{range}(\boldsymbol{Y})} y \cdot \mathbf{Pr}[\boldsymbol{Y} = y] \right) \\ &\qquad \text{(by rearranging terms)} \\ &= \mathbf{E}[\boldsymbol{X}] \cdot \mathbf{E}[\boldsymbol{Y}]. \\ &\qquad \text{(by Definition (Expected value of a random variable))}\end{aligned}$$

Now, we prove the original statement by induction on n. Both sides are identical for the case where $n = 1$, and the above claim is exactly the case where $n = 2$. These are our base cases.

For the inductive case, assume the proposition holds true for $n = k$. We seek to show that it also holds for $n = k + 1$. Suppose $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_{k+1}$ are

independent random variables. Let $\boldsymbol{X} = \prod_{j=1}^{k} \boldsymbol{X}_j$ and $\boldsymbol{Y} = \boldsymbol{X}_{k+1}$. We have

$$\begin{aligned}
\mathbf{E}\left[\prod_{j=1}^{k+1} \boldsymbol{X}_j\right] &= \mathbf{E}[\boldsymbol{X}\boldsymbol{Y}] \\
&= \mathbf{E}[\boldsymbol{X}] \cdot \mathbf{E}[\boldsymbol{Y}] && \text{(by the above claim)} \\
&= \mathbf{E}\left[\prod_{j=1}^{k} \boldsymbol{X}_j\right] \cdot \mathbf{E}[\boldsymbol{X}_{k+1}] \\
&= \left(\prod_{j=1}^{k} \mathbf{E}[\boldsymbol{X}_j]\right) \cdot \mathbf{E}[\boldsymbol{X}_{k+1}] && \text{(by the induction hypothesis)} \\
&= \prod_{j=1}^{k+1} \mathbf{E}[\boldsymbol{X}_j]
\end{aligned}$$

which completes the proof. ■

Exercise 15.21 (Practice with Markov's inequality)**.**
During the Fall 2017 semester, the 15-251 TAs decide to strike because they are not happy with the lack of free food in grading sessions. Without the TA support, the performance of the students in the class drop dramatically. The class average on the first midterm exam is 15%. Using Markov's Inequality, give an upper bound on the fraction of the class that got an A (i.e., at least a 90%) in the exam.

Solution. Let $\boldsymbol{X}$ be the exam score of a student chosen uniformly at random. We will optimistically assume that $\boldsymbol{X}$ is nonnegative. Then $\mathbf{E}[\boldsymbol{X}] = 0.15 \neq 0$ and the fraction of the class that got an A is $\mathbf{Pr}[\boldsymbol{X} \geq 0.9]$. By Markov's inequality,

$$\mathbf{Pr}[\boldsymbol{X} \geq 0.9] \leq \frac{\mathbf{E}[\boldsymbol{X}]}{0.9} = \frac{1}{6}$$

which is an upper bound on the fraction of the class that got an A. ■

Exercise 15.22 (Expectation of a Binomial random variable)**.**
Let $\boldsymbol{X}$ be a random variable with $\boldsymbol{X} \sim \text{Bin}(n, p)$. Determine $\mathbf{E}[\boldsymbol{X}]$ (use linearity of expectation). Also determine $\boldsymbol{X}$'s probability mass function.

Solution. Express $\boldsymbol{X} = \sum_{j=1}^{n} \boldsymbol{X}_j$ where the $\boldsymbol{X}_i$'s are independent and for all i, $\boldsymbol{X}_i \sim \text{Bernoulli}(p)$ as in Definition (Binomial random variable). Note that $\mathbf{E}[\boldsymbol{X}_i] = p$, so by linearity of expectation,

$$\mathbf{E}[\boldsymbol{X}] = \mathbf{E}\left[\sum_{j=1}^{n} \boldsymbol{X}_j\right] = \sum_{j=1}^{n} \mathbf{E}\left[\boldsymbol{X}_j\right] = np.$$

The probability mass function of $\boldsymbol{X}$ is

$$\begin{aligned}
p_{\boldsymbol{X}}(x) &= \mathbf{Pr}[\boldsymbol{X} = x] && \text{(by Definition (Probability mass function))} \\
&= \mathbf{Pr}\left[\sum_{j=1}^{n} \boldsymbol{X}_j = x\right] \\
&= \sum_{J \in \binom{[n]}{x}} \mathbf{Pr}\left[\bigcap_{j \in J} (\boldsymbol{X}_j = 1) \cap \bigcap_{j \notin J} (\boldsymbol{X}_j = 0)\right] \\
& && \text{(by partitioning } \Omega \text{ into } \textstyle\bigcap_{j=1}^{n} (\boldsymbol{X}_j = x_j)) \\
&= \sum_{J \in \binom{[n]}{x}} \left(\prod_{j \in J} \mathbf{Pr}[\boldsymbol{X}_j = 1] \cdot \prod_{j \notin J} \mathbf{Pr}[\boldsymbol{X}_j = 0]\right) \\
& && \text{(by independence of the } X_j\text{'s)} \\
&= \sum_{J \in \binom{[n]}{x}} \left(p^x \cdot (1-p)^{n-x}\right) && \text{(by Definition (Bernoulli random variable))} \\
&= \binom{n}{x} p^x (1-p)^{n-x}.
\end{aligned}$$

■

Exercise 15.23 (Practice with Binomial random variable)**.**
We toss a coin 5 times. What is the probability that we see at least 4 heads?

Solution. Let $\boldsymbol{X}$ be the number of heads among the 5 coin tosses. Then $\boldsymbol{X} \sim$ Binomial$(5, 1/2)$ and so the probability we see at least 4 heads is

$$\mathbf{Pr}[\boldsymbol{X} \geq 4] = \mathbf{Pr}[\boldsymbol{X} = 4] + \mathbf{Pr}[\boldsymbol{X} = 5] = \binom{5}{4}\left(\frac{1}{2}\right)^4\left(\frac{1}{2}\right)^1 + \binom{5}{5}\left(\frac{1}{2}\right)^5\left(\frac{1}{2}\right)^0 = \frac{3}{16}.$$

■

Exercise 15.24 (PMF of a geometric random variable)**.**
Let $\boldsymbol{X}$ be a geometric random variable. Verify that $\sum_{n=1}^{\infty} p_{\boldsymbol{X}}(n) = 1$.

Solution. Recall that for $|r| < 1$, $\sum_{n=0}^{\infty} r^n = 1/(1-r)$. Then

$$\sum_{n=1}^{\infty} p_{\boldsymbol{X}}(n) = \sum_{n=1}^{\infty} (1-p)^{n-1} p = p \cdot \sum_{n=0}^{\infty} (1-p)^n = p \cdot \frac{1}{1-(1-p)} = 1.$$

■

Exercise 15.25 (Practice with geometric random variable)**.**
Suppose we repeatedly flip a coin until we see a heads for the first time. Determine the probability that we flip the coin n times. Determine the expected number of coin flips.

Solution. Let $\boldsymbol{X}$ be the number of flips until we see a head for the first time. Observe that $\boldsymbol{X} = n$ if and only if the first $n-1$ flips land tails and the n-th flip lands heads. This happens with probability exactly

$$\mathbf{Pr}[\boldsymbol{X} = n] = \left(1 - \frac{1}{2}\right)^{n-1} \cdot \frac{1}{2}.$$

Note that this implies $X \sim$ Geometric$(1/2)$. The expected number of coin flips is therefore going to be the expected value of a geometric distribution with parameter $1/2$, which is 2. We will show how to derive this for general p in the next exercise. ■

Exercise 15.26 (Expectation of a geometric random variable)**.**
Let $\boldsymbol{X}$ be a random variable with $\boldsymbol{X} \sim \text{Geometric}(p)$. Determine $\mathbf{E}[\boldsymbol{X}]$.

Solution. Notice that we can extend the previous exercise to see that $\boldsymbol{X}$ models the number of flips of a biased coin (that lands heads with probability p) until the first heads is observed. Let E be the event that the first coin lands heads. Then $\mathbf{Pr}[E] = p$, and $\mathbf{E}[\boldsymbol{X} \mid E] = 1$ since if the first coin lands heads $\boldsymbol{X} = 1$ with probability 1.

If the first coin lands tails, this does not affect the number of future flips before we see the first heads. In other words, we are at the same state as we were before flipping the first coin. Thus, we can say that $\mathbf{E}[\boldsymbol{X} \mid \overline{E}] = 1 + \mathbf{E}[\boldsymbol{X}]$. This is called the *memorylessness property* of the geometric distribution.

Now, by the law of total expectation, we can say that

$$\mathbf{E}[\boldsymbol{X}] = \mathbf{E}[\boldsymbol{X} \mid E] \cdot \mathbf{Pr}[E] + \mathbf{E}[\boldsymbol{X} \mid \overline{E}] \cdot \mathbf{Pr}[\overline{E}] = 1 \cdot p + (1 + \mathbf{E}[\boldsymbol{X}]) \cdot (1 - p).$$

Solving for $\mathbf{E}[\boldsymbol{X}]$, we get that $\mathbf{E}[\boldsymbol{X}] = 1/p$. ■

Chapter 16

Randomized Algorithms

Exercise 16.1 (Las Vegas to Monte Carlo)**.**
Suppose you are given a Las Vegas algorithm A that solves $f : \Sigma^* \to \Sigma^*$ in expected time $T(n)$. Show that for any constant $\epsilon > 0$, there is a Monte Carlo algorithm that solves f in time $O(T(n))$ and error probability ϵ.

Solution. Given the Las Vegas algorithm A and a constant $\epsilon > 0$, we construct a Monte Carlo algorithm A' with the desired properties as follows.

```
x: string.
A'(x):
1 Run A(x) for (1/ε)T(|x|) steps.
2 If A terminates, return its output.
3 Else, return "failure".
```

Since ϵ is a constant, the running time of A' is $O(T(n))$. The error probability of the algorithm can be bounded using Theorem (Markov's inequality) as follows. For any $x \in \Sigma^*$, let $\boldsymbol{T}_x$ be the random variable that denotes the number of steps $A(x)$ takes. Note that by Definition (Las Vegas algorithm), $\mathbf{E}[\boldsymbol{T}_x] \leq T(|x|)$ for all x. In the event of $A'(x)$ failing, it must be the case that $\boldsymbol{T}_x > \frac{1}{\epsilon}T(|x|)$. So the probability that $A'(x)$ fails can be upper bounded by

$$\mathbf{Pr}\left[\boldsymbol{T}_x > \frac{1}{\epsilon}T(|x|)\right] \leq \mathbf{Pr}\left[\boldsymbol{T}_x \geq \frac{1}{\epsilon}\mathbf{E}[\boldsymbol{T}_x]\right] \leq \epsilon,$$

where the last inequality follows from Markov's Inequality.

Technicality: There is a small technical issue here. Algorithm A' needs to be able to compute $T(|x|)$ from x in $O(T(|x|))$ time. This is indeed the case for most $T(\cdot)$ that we care about. ■

Exercise 16.2 (Monte Carlo to Las Vegas)**.**
Suppose you are given a Monte Carlo algorithm A that runs in worst-case $T_1(n)$ time and solves $f : \Sigma^* \to \Sigma^*$ with success probability at least p (i.e., for every input, the algorithm gives the correct answer with probability at least p and takes at most $T_1(n)$ steps). Suppose it is possible to check in $T_2(n)$ time whether the output produced by A is correct or not. Show how to convert A into a Las Vegas algorithm that runs in expected time $O((T_1(n) + T_2(n))/p)$.

Solution. Given the Monte Carlo algorithm A as described in the question, we create a Las Vegas algorithm A' as follows.

```
x: string.
A'(x):
1 Repeat:
2     Run A(x).
3     If the output is correct, return the output.
```

For all $x \in \Sigma^*$, the algorithm gives the correct answer with probability 1.

For $x \in \Sigma^*$, define $\boldsymbol{T}_x$ to be the random variable corresponding to the number of iterations of the above algorithm. Observe that $\boldsymbol{T}_x$ is a geometric random variable (Definition (Geometric random variable)) with success probability p (i.e., $\boldsymbol{T}_x \sim$ Geometric(p)). The total number of steps $A'(x)$ takes is thus $(T_1(|x|) + T_2(|x|)) \cdot \boldsymbol{T}_x$ (ignoring constant factors). The expectation of this value is $(T_1(|x|) + T_2(|x|)) \cdot \mathbf{E}[\boldsymbol{T}_x]$, where $\mathbf{E}[\boldsymbol{T}_x] = 1/p$. So the total expected running time is $O((T_1(|x|) + T_2(|x|))/p)$. ■

Exercise 16.3 (Boosting for one-sided error)**.**
This question asks you to boost the success probability of a Monte Carlo algorithm computing a decision problem with *one-sided error*.

Let $f : \Sigma^* \to \{0,1\}$ be a decision problem, and let A be a Monte Carlo algorithm for f such that if x is a YES instance, then A always gives the correct answer, and if x is a NO instance, then A gives the correct answer with probability at least $1/2$. Suppose A runs in worst-case $O(T(n))$ time. Design a new Monte Carlo algorithm A' for f that runs in $O(nT(n))$ time and has error probability at most $1/2^n$.

Solution. Here is the description of A'.

```
x: string.
A'(x):
1 Repeat |x| times:
2    Run A(x).
3    If it outputs 0, output 0.
4 Output 1.
```

We call $A(x)$ n times, and the running time of A is $O(T(n))$, so the overall running time of A' is $O(nT(n))$.

To prove the required correctness guarantee, we need to show that for all inputs x, $Pr[A'(x) \neq f(x)] \leq 1/2^n$. For any x such that $f(x) = 1$, we know that $\mathbf{Pr}[A(x) = 1] = 1$, and therefore $\mathbf{Pr}[A'(x) = 1] = 1$. For any x such that $f(x) = 0$, we know that $\mathbf{Pr}[A(x) = 0] \geq 1/2$. The only way A' makes an error in this case is if $A(x)$ returns 1 in each of the n iterations. So if E_i is the event that in iteration i, $A(x)$ returns the wrong answer (i.e. returns 1), we are interested in upper bounding $\mathbf{Pr}[\text{error}] = \mathbf{Pr}[E_1 \cap E_2 \cap \cdots \cap E_n]$. Note that the E_i's are independent (one run of $A(x)$ has no effect on other runs of $A(x)$). Furthermore, for all i, $\mathbf{Pr}[E_i] \leq 1/2$. So

$$\mathbf{Pr}[E_1 \cap E_2 \cap \cdots \cap E_n] = \mathbf{Pr}[E_1]\,\mathbf{Pr}[E_2] \cdots \mathbf{Pr}[E_n] = 1/2^n,$$

as desired. ■

Exercise 16.4 (Boosting for two-sided error)**.**
This question asks you to boost the success probability of a Monte Carlo algorithm computing a decision problem with *two-sided error*.

Let $f : \Sigma^* \to \{0,1\}$ be a decision problem, and let A be a Monte Carlo algorithm for f with error probability $1/4$, i.e., for all $x \in \Sigma^*$, $\mathbf{Pr}[A(x) \neq f(x)] \leq 1/4$. We want to boost the success probability to $1 - 1/2^n$, and our strategy will be as follows. Given x, run $A(x)$ $6n$ times (where $n = |x|$), and output the more common output bit among the $6n$ output bits (breaking ties arbitrarily). Show that the probability of outputting the wrong answer is at most $1/2^n$.

Solution. Let $\boldsymbol{X}_i$ be a Bernoulli random variable corresponding to whether the algorithm gives the correct answer in iteration i. That is,

$$\boldsymbol{X}_i = \begin{cases} 1 & \text{if algorithm gives correct answer in iteration } i, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{X} = \sum_{i=1}^{6n} \boldsymbol{X}_i$. So $\boldsymbol{X} \sim \text{Bin}(6n, 3/4)$ (see Definition (Binomial random variable)). Note that

$$\mathbf{Pr}[\boldsymbol{X} = i] = \binom{6n}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{6n-i}.$$

Furthermore, $\mathbf{Pr}[\text{error}] \leq \mathbf{Pr}[\boldsymbol{X} \leq 3n]$, and so

$$\mathbf{Pr}[\text{error}] \leq \sum_{i=0}^{3n} \mathbf{Pr}[\boldsymbol{X} = i] = \sum_{i=0}^{3n} \binom{6n}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{6n-i}.$$

We simplify the right-hand-side as follows.

$$\begin{aligned}
\sum_{i=0}^{3n} \binom{6n}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{6n-i} &= \sum_{i=0}^{3n} \binom{6n}{i} \frac{3^i}{4^{6n}} \\
&\leq \sum_{i=0}^{3n} \binom{6n}{i} \frac{3^{3n}}{4^{6n}} \\
&= \frac{3^{3n}}{4^{6n}} \cdot \sum_{i=0}^{3n} \binom{6n}{i} \\
&= \frac{3^{3n}}{4^{6n}} \cdot 2^{6n} \\
&= \frac{27^n}{64^n} < \frac{1}{2^n}.
\end{aligned}$$

■

Exercise 16.5 (Maximum number of minimum cuts)**.**
Using the analysis of the randomized minimum cut algorithm, show that a graph can have at most $n(n-1)/2$ distinct minimum cuts.

Solution. Suppose there are t distinct minimum cuts $F_1, F_2, \ldots F_t$. Our goal is to show $t \leq \binom{n}{2}$. Let A_i be the event that the first phase of our algorithm in the proof of Theorem (Contraction algorithm for min cut) outputs F_i (the first phase refers to the phase before the boosting). We know that for any i, $\mathbf{Pr}[A_i] \geq 1/\binom{n}{2}$ (as shown in the proof). Furthermore, the events A_i are disjoint (if one happens, another cannot happen). So

$$\mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_t] = \mathbf{Pr}[A_1] + \mathbf{Pr}[A_2] + \cdots + \mathbf{Pr}[A_t] \geq \frac{t}{\binom{n}{2}}.$$

Since $\mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_t] \leq 1$, we can conclude

$$t \leq \binom{n}{2}.$$

■

Exercise 16.6 (Contracting two random vertices)**.**
Suppose we modify the min-cut algorithm seen in class so that rather than picking an edge uniformly at random, we pick 2 vertices uniformly at random and contract them into a single vertex. True or False: The success probability of the algorithm (excluding the part that boosts the success probability) is $1/n^k$ for some constant k, where n is the number of vertices. Justify your answer.

Solution. Let A and B be cliques of size $n/2$ each. Join them together by a single edge to form the graph G. Then the minimum cut is $S = A$ with the single edge connecting A and B being the cut edge. Observe that the algorithm will output this cut if and only if it never picks vertices $a \in A$ and $b \in B$ to contract. The probability that the algorithm never picks $a \in A$ and $b \in B$ to contract is exponentially small. (We leave this part to the reader. Note that all you need is a bound; you do not have to calculate the probability exactly.) ■