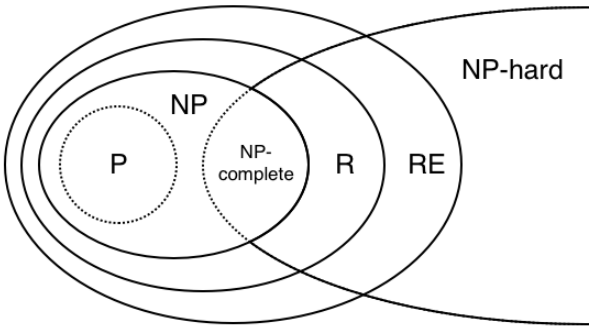
CS-251 (EPFL)
Theory of Computation

David Duan

Spring 2020

This course constitutes an introduction to theory of computation. It discusses the basic theoretical models of computing as well as provides a solid and mathematically precise understanding of their fundamental capabilities and limitations. Topics include

- Basic models of computation (finite automata, Turing machine);
- Elements of computability theory (undecidability, reducibility);
- Introduction to time complexity theory (P, NP and theory of NP-completeness).



Overview of CS-251.

- **P**: polynomial-time;
- **NP**: non-deterministic polynomial time;
- **R**: decidable languages;
- **RE**: recognizable languages

Course Overview

What are the fundamental capabilities and limitations of computers? In this course, we look at the three traditionally central ares of the theory of computation:

- **Automata theory** - Basic definitions and properties of mathematical models of computation.
- **Computability theory** - Certain basic problems cannot be solved by computers...
- **Complexity theory** - What makes some problems computationally hard and others easy?

1. Automata Theory

A **finite-state machine** (**FSM**) or **finite automaton** is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some ***inputs***; the change from one state to another is called a ***transition***. Note the FSM has less computational power than some other models of computation such as the Turing machine. This is mainly because an FSM's memory is limited by the number of states it has.

1.1. Deterministic Finite Automaton

A **deterministic finite automaton** (**DFA**) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of

- a finite set of ***states*** Q;
- a finite set of input symbols called the ***alphabet*** Σ;
- a ***transition function*** $\delta : Q \times \Sigma \to Q$, i.e., (current state, symbol) $\mapsto$ next state;
- an initial or ***start state*** $q_0 \in Q$;
- a set of ***accept states*** $F \subseteq Q$.

Let $w = a_1 a_2 \dots a_n$ be a string over the alphabet Σ. The DFA M **accepts** the string w if a sequence of states $r_0, r_1, \dots, r_n$ exists in Q satisfying the following conditions:

1. $r_0 = q_0$ (we start at the initial state);
2. $r_{i+1} = \delta(r_i, a_{i+1}$ for $i = 0, \dots, n-1$ (we follow the transition function);
3. $r_n \in F$ (we halt in one of the accept states).

Otherwise, we say the automaton M **rejects** the string. The set of strings that M accepts is the language **recognized** by M and this language is denoted by $L(M)$.

1.2. Non-deterministic Finite Automaton (with ε-transitions)

A **non-deterministic finite automaton** (**NFA**) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, consisting of

- a set of states Q, alphabet Σ, initial state q_0, a set of accept states F;
- a ***transition function*** $\delta : Q \times \{\Sigma \cup \{\varepsilon\}\} \to 2^Q$ (output is a set of states, hence the power set).

To summarize, NFAs have the following characteristics that differentiate them from DFAs:

- If an arrow is labeled ε, then it can be taken *without* reading any input symbol.
- A state may have 0, 1, or many exiting arrows (transitions) for each input symbol.
- When a state has no transitions on a symbol, the NFA halts that branch of execution.

1.3. From NFA to DFA (Subset Construction)

For every NFA N, there exists a DFA M such that $L(M) = L(N)$. Given NFA $(Q_N, \Sigma, \delta_N, q_0, F_N)$, we can construct an equivalent DFA $M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ as follows.

- $Q_M = 2^{Q_N}$: states of M correspond to subsets of Q_N;
- $\delta_M(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$, where $E(R)$ is the set of states that can be reached from R by travelling along 0 or more ε-transitions;
- $q_M = \{q_N\}$: M starts in the state corresponding to the set containing just the state of N;
- $F_M = \{R \in Q_M \mid R$ contains an accept state of $N\}$: the machine M accepts if one of the possible states that N could be at this point is an accept state.

1.4. Regular Languages

A **regular language** is a formal language that can be expressed using a *regular expression*. Alternatively, it can be defined as a language recognized by a DFA/NFA. The equivalence of regular expressions and finite automata is known as **Kleene's theorem**.

We define the **regular operations** on languages A and B as follows:

- **Complement**: $\overline{A} := \{x \mid x \notin A\}$;
- **Union**: $A \cup B := \{x \mid x \in A \vee x \in B\}$;
- **Intersection**: $A \cap B := \{x \mid x \in A \wedge x \in B\}$;
- **Concatenation**: $A \circ B = \{xy \mid x \in A \wedge y \in B\}$;
- **(Kleene) Star**: $A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, \forall i : x_i \in A\}$;

The class of regular languages is closed under all five operations above. This can be shown by explicitly constructing an appropriate DFA/NFA given the DFA recognizing languages A and B.

1.5. Pumping Lemma

The **pumping lemma** for regular languages is a lemma that describes an essential property of all regular languages. Informally, it says that all sufficiently long words in a regular language may be *pumped*–that is, have a middle section of the word repeated an arbitrary number of times–to produce a new word that also lies in the same language. It is useful for disproving the regularity of a specific language in question.

Theorem. *If A is a regular language, then there exists a number $p \in \mathbb{Z}$, called the **pumping length**, such that for any $s \in A$ of length at least p, there exists a division of $s = xyz$ such that $xy^i z \in A$ for all $i \geq 0$, $|y| > 0$, and $|xy| \leq p$.*

2. Computability Theory

The **Church-Turing Thesis** states that the intuitive notion of algorithms is equivalent to Turing machine algorithms.

2.1. Turing Machines

A **Turing machine** (**TM**) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q0, q_{\text{accept}}, q_{\text{reject}})$ where Q, Σ, Γ are all finite sets and

1. Q is the set of ***states***,
2. Σ is the ***input alphabet*** not containing the **blank symbol** $\sqcup$,
3. Γ is the ***tape alphabet***, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times T \to Q \times \Gamma \times \{L, R\}$ is the ***transition function***,*
5. $q_0 \in Q$ is the ***start state***,
6. $q_{\text{accept}} \in Q$ is the ***accept state***, and
7. $q_{\text{reject}} \in Q$ is the ***reject state***, where $q_{\text{reject}} \neq q_{\text{accept}}$.

We can represent various computation problems by languages. For example, the *acceptance problem* for DFAs for testing whether a particular DFA accepts a given string can be expressed as a language, A_{DFA}, which contains the encoding of all DFAs together with strings that the DFAs accept. From now on, let $\langle \cdot \rangle$ denote the binary representation of the argument.

2.2. Turing-Decidable Languages

A TM M **decides** a language $L \subseteq \Sigma^*$ iff for all inputs $w \in \Sigma^*$:

- If $w \in L$, M accepts w.
- If $w \notin L$, M rejects w.

We say M is a **decider** for L and L is **Turing-decidable**. Examples of decidable languages include

- (Acceptance testing of DFA) $A_{\text{DFA}} := \{\langle A, w \rangle \mid A \text{ is a DFA and } w \in L(A)\}$.
- (Emptiness testing of DFA) $E_{\text{DFA}} := \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \varnothing\}$.
- (Equivalence testing of DFA) $EQ_{\text{DFA}} := \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$.

2.3. Turing-Recognizable Languages

A TM M **recognizes** a language $L \subseteq \Sigma^*$ iff for all inputs $w \in \Sigma^*$:

- If $w \in L$, M accepts w.
- If $w \notin L$, M either rejects w or *never halts*.

We say M is a **recognizer** for L and L is **Turing-recognizable**. Examples include

- (Acceptance testing of TM) $A_{\text{TM}} := \{\langle M, w \rangle \mid M \text{ is a TM and } w \in L(M)\}$.
- (Emptiness testing of TM) $E_{\text{TM}} := \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \varnothing\}$.†
- (The halting problem) $HALT_{\text{TM}} := \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$.

We can use the *diagonalization method* to show that some languages are not Turing-recognizable.

* Suppose the machine is at state q and the head is over a tape square containing a symbol a. If $\delta(q, a) = (r, b, L)$, the machine writes the symbol b replacing the a at the current square, goes to state r, and move to the left after writing.

† In fact, a general result, called **Rice's theorem**, states that determining any non-trivial property of the languages recognized by TMs is undecidable. See appendix for more information.

2.4. Co-Turing-Recognizable Languages and Turing-Unrecognizable Languages

A language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

Theorem. *A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.*

Proof. Suppose A is Turing-decidable. Then it is Turing-recognizable, and a complement of a decidable language also is decidable and hence recognizable. Now suppose both A and $\overline{A}$ are Turing-recognizable. Let M_1 be the recognizer for A and M_2 be the recognizer for $\overline{A}$. The following TM M is a decider for A: On input w,

1. Run both M_1 and M_2 on input w in parallel.
2. If M_1 accepts, accept; if M_2 accepts, reject.

Since every string w is either in A or $\overline{A}$, either M_1 or M_2 must accept w. Thus, M will halt, accepts all strings in A, and rejects all strings not in A. It follows that M is a decider for A and A is decidable. □

Since both A_{TM} and $HALT_{\text{TM}}$ are recognizable but undecidable, their complements $\overline{A}_{\text{TM}}$ and $\overline{HALT}_{\text{TM}}$ are unrecognizable. Other examples include EQ_{TM} and $\overline{EQ}_{\text{TM}}$.

2.5. Closure Properties of Decidable and Recognizable Languages

It is worth mentioning that both decidable and recognizable languages are closed under union, intersection, concatenation, and star. Decidable languages are closed under complementation but recognizable languages are not.

2.6. Reducibility

A **reduction** is a way of converting one problem to another problem such that a solution to the second problem can be used to solve the first problem. It is critical for us to prove decidability/recognizability/unrecognizability of a language. We introduce mapping reducibility here and leave polynomial-time mapping reducibility to the next section.

A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **computable function** if there is some TM M with the following behavior: On input w, compute $f(w)$ and write it on the tape; move the tape head to the start of $f(w)$; halt.

A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **mapping reduction** from A to B, written $A \leq_m B$, iff

- $\forall w \in \Sigma_1^* : w \in A \iff f(w) \in B$;
- f is a computable function.

Intuitively, $A \leq_m B$ means:

- A is not harder than B, i.e., if B is decidable/recognizable/co-recognizable, then A is decidable/recognizable/co-recognizable.
- B is at least as hard as A, i.e., if A is undecidable/unrecognizable/not co-recognizable, then B is undecidable/unrecognizable/not co-recognizable.

Note that $\leq_m$ is a transitive relation and $A \leq_m B \implies \overline{A} \leq_m \overline{B}$.

3. Complexity Theory

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The **time complexity class**, denoted TIME(t(n)), is defined be the collection of all languages that are decidable by an $O(t(n))$ time TM.

3.1. The Class P

P (stands for **polynomial time**) is the class of languages that are decidable in polynomial time on a deterministic TM. In other words,

$$\mathbf{P} = \bigcup_k \text{TIME}(n^k).$$

3.2. The Class NP

A **verifier** for a language A is an algorithm V where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

NP (stands for **non-deterministic polynomial time**) is the class of languages that have polynomial time verifiers.

3.3. Polynomial Time Reducibility

A **polynomial time computable function** is a computable function where the TM M has a polynomial time complexity. A function $f : \Sigma_1^* \to \Sigma_2^*$ is called a **polynomial time mapping reduction** from A to B, written $A \leq_P B$, iff

- $\forall w \in \Sigma_1^* : w \in A \iff f(w) \in B$;
- f is a polynomial time computable function.

Theorem. If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.

3.4. NP-Completeness

The decision problem X is **NP-hard** if every problem $A \in \mathbf{NP}$ satisfies $A \leq_P X$. The decision problem X is **NP-complete** if every problem $A \in \mathbf{NP}$ satisfies $A \leq_P X$.

Theorem. *Let X be any* **NP***-complete problem. Then:*

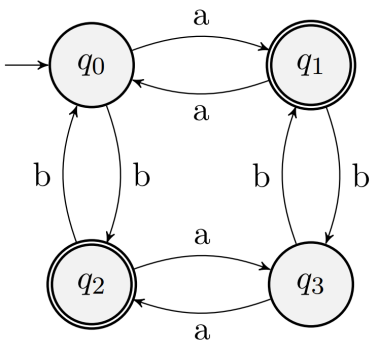
- *If $X \in \mathbf{P}$, then* $\mathbf{P} = \mathbf{NP}$ *and so every problem in* **NP** *can be solved with a polynomial-time algorithm.*
- *If $X \notin \mathbf{P}$, then* $\mathbf{P} \neq \mathbf{NP}$ *and no* **NP***-complete problem can be solved with a polynomial-time algorithm.*

As a result, proving that a problem is **NP**-complete is a very strong indication that a problem cannot be solved efficiently.

A. Appendix: Automata

A.1. Language Recognized by a DFA

Example. For the DFA given below find a description of the language it recognizes. Prove that your description is correct using induction.



Solution. The language recognized by the automaton $L(M)$ is

$$L = \{w \mid w \in \{a, b\}^* \wedge |w| \text{ is odd}\}.$$

Let x be the input string of length $|x| = l$. We claim that

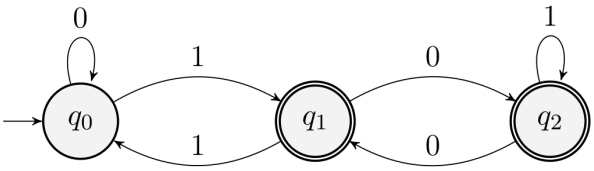
- if x contains an even number of a's and an even number of b's, M finishes in q_0;
- if x contains an odd number of a's and an even number of b's, M finishes in q_1;
- if x contains an even number of a's and an odd number of b's, M finishes in q_2;
- if x contains an odd number of a's and an odd number of b's, M finishes in q_3.

We prove the claim by induction on l, the length of x. First, if $l = 0$, then x is the empty string which contains an even number of a's and an even number of b's. Indeed, M finishes in q_0. Now suppose the claim is true for all $l < n$ for some $n \in \mathbb{Z}^+$. Let $l = n$ and let x' be the first $n - 1$ digits of x. Since $|x'| < n$ the induction hypothesis applies. We have four cases.

Suppose x' contains an even number of a's and an even number of b's (so one step before the last by the hypothesis we are at q_0) and consider the last input digit. If the input is now a, then x contains an odd number of a's and an even number of b's and indeed M transitions to q_1 and finishes. If the input is now b, then x contains an even number of a's and an odd number of b's and indeed M transitions to q_2 and finishes. (The other cases are similar and thus omitted.)

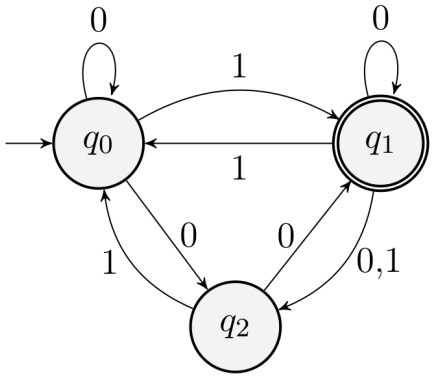
By induction, the hypothesis holds for $l = n$ and this completes the proof. □

Practice. Describe the language recognized by the following DFA and prove the correctness of your claim. (Hint: this DFA recognizes all strings of binary numbers that are not divisible by 3.)



A.2. From NFA to DFA

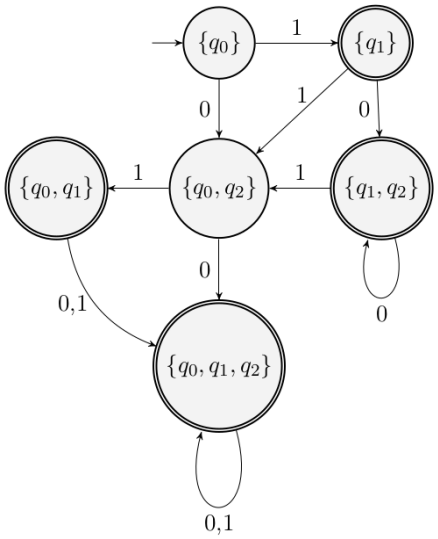
Example. Construct a DFA that accepts the same language as that accepted by the NFA below.



Solution. Given $Q_N = \{q_0, q_1, q_2\}$, the set of states for the equivalent DFA M has cardinality $2^{|Q_N|}$ with start state $\{q_0\}$. The transition function δ_M is described below:

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0, q_2\}$	$\{q_1\}$
$\{q_1\}$	$\{q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_2\}$	$\{q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Observe the states $\varnothing$ and $\{q_2\}$ are unreachable from the start state $\{q_0\}$. Thus, we can remove them from the automaton. The accepting states of M are all the states corresponding to a subset of Q_N containing an accepting state of the original NFA: $\{q_1\}, \{q_0, q_1\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}$. The final automaton is given below:

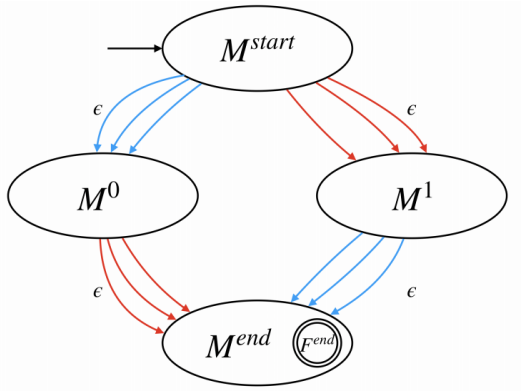


A.3. Regular Languages

Example. Let $L \subseteq \{0,1\}^*$ be a language. Define DROP-OUT(L) to be the language containing all strings that can be obtained by removing one 0 and one 1 from a string in L (in any order), i.e.,

$$\text{DROP-OUT}(L) = \{xyz \mid x0y1z \in L \text{ or } x1y0z \in L, \text{ where } x, y, z, \in \{0,1\}^*\}.$$

Prove that if L is regular, then so is DROP-OUT(L).



Solution. Since L is regular, there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts it. To prove that DROP-OUT(L) is also regular, it suffices to create an NFA that accepts it. For this, we construct four copies of M: $M^{\text{start}}, M^0, M^1, M^{\text{end}}$. We add the superscript to each component of the DFA tuple accordingly. For each transition in M^{start} that reads a 0, i.e., $\delta^{\text{start}}(s_i^{\text{start}}, 0) = s_j^{\text{start}}$, we add a new ε-transition from s_i^{start} to s_j^0 and a new ε-transition from s_i^1 to s_j^{end}. Similarly, for each 1-transition in M^{start}, we add new ε-transitions from M^{start} to M^1 and from M^0 to M^{end}. We now give the technical definition of the NFA $(Q^N, \Sigma^N, \delta^N, q_0^N, F^N)$:

- $Q^N = Q^{\text{start}} \cup Q^0 \cup Q^1 \cup Q^{\text{end}}$; $q_0^N = q_0^{\text{start}}$; $\Sigma^N = \Sigma^{\text{start}}$; $F^N = F^{\text{end}}$.
- Transition in the copies: $\forall q^\pi \in Q^\pi, \delta^N(q^\pi, \alpha) = \delta^\pi(q^\pi, \alpha), \pi \in \{\text{start}, 0, 1, \text{end}\}$.
- New transitions:
  - $\forall q_i^{\text{start}}, q_j^{\text{start}} \in Q^{\text{start}} : \delta^{\text{start}}(q_i^{\text{start}}, 0) = q_j^{\text{start}}$, set $\delta^N(q_i^{\text{start}}, \varepsilon) = q_j^0$, $\delta^N(q_i^1, \varepsilon) = q_j^{\text{end}}$.
  - $\forall q_i^{\text{start}}, q_j^{\text{start}} \in Q^{\text{start}} : \delta^{\text{start}}(q_i^{\text{start}}, 1) = q_j^{\text{start}}$, set $\delta^N(q_i^{\text{start}}, \varepsilon) = q_j^1$, $\delta^N(q_i^0, \varepsilon) = q_j^{\text{end}}$.

To prove the correctness of this construction, we need to show that $xyz \in$ DROP-OUT(L) iff $x0y1z \in L$ or $x1y0z \in L$. Let xyz be a string such that $x0y1z \in L$. By construction, after reading x we can take an ε-transition from M^{start} to M^0; after reading y in M^0 we can take an ε-transition to M^{end}. Finally, after reading z in M^{end}, we will arrive at an accepting state of M^{end}. Thus, xyz is accepted by N and $xyz \in$ DROP-OUT(L). A mirror proof shows that xyz will be accepted by N if $x1y0z \in L$. Conversely, let w be a string accepted by N. Since the starting state is in M^{start} and all accepting states are in M^{end}, we must have taken two ε-transitions at some point that correspond to avoiding reading one 0 and one 1 in some order. Thus w can be written in the form of xyz where xyz are the three parts of w separated by the two ε-transitions, and we have $x0y1z \in L$ or $x1y0z \in L$. □

Practice. Given a language $L \subseteq \{0,1\}^*$, define $L_{11} = \{xy \mid x11y \in L\}$. In words, each string in L_11 is obtained by taking a string in L and deleting two consecutive 1s from it. Prove that if L is regular, so is L_{11}. (Hint: Define an NFA with transition rules that pretend we have read two ones, i.e., $\forall q_i, q_j, q_k \in Q : \delta(q_i, 1) = q_j \wedge \delta(q_j, 1) = q_k$, add $\delta(q_i, \varepsilon) = q_k$.)

A.4. Pumping Lemma

Exercise 1. Prove the language $A = \{0^{n^2}1^n \mid n \geq 0\}$ is not regular.

Solution 1. We use the pumping lemma. Suppose that A is regular with pumping length p. For $s := p^{p^2}1^p \in A$, by the pumping lemma, s can be decomposed as $s = xyz$ where $|xy| \leq p$, $|y| \geq 1$, and $xy^iz \in A$ for all $i \in \mathbb{N}$. Since s starts with $p^2 \geq p$ zeros, $|xy| \leq p$ and $|y| \geq 1$, we have $y = 0^k$ for some integer $k \geq 1$. Then $xy^2z = 0^{p^2+k}1^p \in A$ by the pumping lemma, a contradiction. Hence, A is not regular. □

Exercise 2. Prove the language $L = \{0^m1^m : n, m \in \mathbb{N}, n \leq m \vee n \geq 2m\}$ is not regular.

Solution 2. We use the pumping lemma. Suppose that L is regular with pumping length p. By taking $n = m = p + 1$, we have $s := 0^{p+1}1^{p+1} \in A$. By the pumping lemma, s can be decomposed as $s = xyz$ where $|xy| \leq p$, $|y| \geq 1$, and $xy^iz \in A$ for all $i \in \mathbb{N}$. Since s starts with $p + 1$ zeros, $y = 0^k$ for some integer k, $1 \leq k \leq p$. Then $xy^2z = 0^{p+1+k}1^{p+1} \in A$ by the pumping lemma. However, $p + 1 < p + 1 + k < 2(p + 1)$ for all $1 \leq k \leq p$, which contradicts that $xy^2z \in A$. Hence, L is not regular. □

Exercise 3. Prove the language $C = \{w \mid w \text{ has equal number of 0s and 1s}\}$ is not regular.

Solution 3. We can prove it by contradiction by intersecting it with the regular language $\{0^*1^*\}$ hence yielding $B = \{0^n1^n \mid n \geq 0\}$ which we know is not regular. Since the intersection of two regular languages must be regular but B is not, we know C is not regular. □

Exercise 4. Prove the language $A = \{0^{2^n} : n \geq 0\}$ is not regular.

Solution 4. Suppose A is regular and let is pumping length be p. Let 2^r be any power of two strictly greater than p. Consider the string $s = 0^{2^r} \in A$. By the pumping lemma s can be written as xyz where $|xy| \leq p, |y| \geq 1, xy^iz \in A$ for any $i \geq 0$. Since s contains only 0's, y must be 0^k for some $1 \leq k \leq p$. Therefore, $xy^2z = 0^{2^r+k}$ where the number of zeros is more than 2^r but less than 2^{r+1}, since $2^r + k \leq 2^r + p < 2^r + r^2 = 2^{r+1}$. Thus $xy^2z \notin A$. Contradiction. □

Practice 1. Prove the language $B = \{0^n1^n \mid n \geq 0\}$ is not regular. (Hint: consider 0^p1^p.)

Practice 2. Prove the language $F = \{ww \mid w \in \{0, 1\}^*\}$ is not regular. (Hint: consider $w = 0^p1^p$.)

Practice 3. Let s^R be the reverse of the string s. Prove the language $K = \{ss^R \mid s \in \{0, 1\}^*\}$ is not regular. (Hint: consider the string $s = 0^p110^p$.)

Practice 4. Prove the language $E = \{0^i1^j \mid i > j\}$ is not regular. (Hint: consider $s = 0^{p+1}1^p$ and xz.)

Practice 5. Prove the language $F = \{ww \mid w \in \{0, 1\}^*\}$ is not regular. (Hint: consider $w = 0^p1^p$.)

B. Appendix: Computability

B.1. Prove decidability

To show a language is decidable, thanks to Church-Turing Thesis, it suffcies to simply providing a high-level algorithm (in terms of TMs, of course) that decides the problem.

Example. Prove the following language is decidable.

$$A = \{\langle D \rangle \mid D \text{ is a DFA over } \{0,1\} \text{ which accepts everything except } w \text{ for some } w \in \{0,1\}^*\}$$

Solution. Given $\bar{D}$, find a string w accepted by $\bar{D}$ by finding a path from the start of $\bar{D}$ to an accepting state. If $\bar{D}$ has no such path/string, then it does not accept anything, so $\langle \bar{D} \rangle$ is not in A. Next, construct a DFA M that recognizes all strings except w. Finally, verify that the DFA $M \cap \bar{D}$ has no path from the start state to the accepting state. In other words, $M \cap \bar{D}$ should have no accepting strings. If so then $\langle \bar{D} \rangle \in A$. Otherwise $\langle \bar{D} \rangle \notin A$. □

B.2. Prove undecidability

To show a language L is undecidable, show that if there exists a decider for L, then one can build a decider S for A_{TM} (or any other undecidable problems, e.g., $HALT_{\text{TM}}$). Since S cannot exist, L must be undecidable.

Example. Prove the following language is undecidable: $E_{\text{TM}} = \{\langle M \rangle \mid L(M) = \varnothing\}$.
Solution. Suppose that R is a decider for E_{TM}. We construct a decider S for A_{TM} as follows.

1. Given $\langle T, w \rangle$, build a TM $\langle M \rangle$ as follows.
   - Ignore its input.
   - Simulate T on w and return what T would have returned on w.
2. Simulate R with $\langle M \rangle$ as input.
3. If R accepts, then reject. Otherwise accept.

To show that S decides A_{TM}, observe if $\langle T, w \rangle \in A_{\text{TM}}$, then M (built by S for the input $\langle T, w \rangle$) accepts all strings, i.e., $L(M) \neq \varnothing$. Thus, R accepts and S rejects. It follows that $\langle T, w \rangle \in L(S)$. A mirror argument shows that if $\langle T, w \rangle \notin A_{\text{TM}}$ then $\langle T, w \rangle \notin L(S)$. Furthermore, S always halts as R always halts. Thus, S decides A_{TM}. Contradiction. □

Practice. Prove the following language is undecidable: $K = \{\langle M \rangle \mid |L(M)| < \infty\}$. (Hint: the same proof works, because M either accepts all strings or no string (it ignores the input); therefore, $L(M)$ is finite iff it is empty.)

B.3. Prove recognizability

Example 1. Prove the following language is recognizable.

$$L = \{\langle M \rangle \mid \ M \text{ is a TM and } M \text{ accepts at least one string } w \in \{0\}^* \setminus \{\varepsilon\}\}.$$

Solution 1. Consider the following procedure: S = on input $\langle M \rangle$:

1. For every positive integer l and every string $w \in \{0\}^* \setminus \{\varepsilon\}$ with $|w| \leq l$,
2. Simulate M on w for l steps.
3. If M accepts w (during l steps), accept $\langle M \rangle$.

We can prove that S is a recognizer for L_2. If $\langle M \langle \in L_2$, then there exists a string $w_0 \in \{0\}^* \setminus \{\varepsilon\}$ accepted by M. If M accepts w_0 in l_0 steps, then S accepts $\langle M \rangle$ before l becomes $l_0 + 1$. Otherwise S will not terminate when run on $\langle M \rangle$. Hence, $L(S) = L$. □

Example 2. Show that following language is recognizable but undecidable.

$$L = \{\langle M, D \rangle \mid M \text{ is a TM, } D \text{ is a DFA, } L(D) \cap L(M) \neq \varnothing\}.$$

Solution 2. We first build a recognizer for L. Given input $\langle M, D \rangle$,

1. For $i = 0, 1, \ldots,$
2. For $w \in \Sigma^*$ such that $|w| \leq i$:
3. Run D and M on w for i steps.
4. If both accept, return accept.

All simulations are finite, so the algorithm eventually reaches every value of i. If $\langle M, D \rangle \in L$, then there exists a $w \in L(M) \cap L(D)$. If $|w| = i$ and is accepted by M steps and by D in at most j steps, the recognizer will accept $\langle M, D \rangle$ when $i = \max(j, k)$.

We now construct a reduction $A_{\text{TM}} \leq_m L$: $f : \{\langle T, w \rangle\} \mapsto \{\langle M, D \rangle\}$.

- D accepts any string.
- M ignores its input and simulates T on w and outputs the result.

Then $L(D) = \Sigma^*$. If $\langle T, w \rangle \in A_{\text{TM}}$, then $L(M) \cap L(D) = \Sigma^*$ and $\langle M, D \rangle \in L$. If $\langle T, w \rangle \notin A_{\text{TM}}$, $L(M) \cap L(D) = \varnothing$ and $\langle M, D \rangle \notin L$. □

B.4. Prove unrecognizability

To prove a language is unrecognizable, construct a mapping reduction from $\overline{A}_{\text{TM}}$ (or any other known unrecognizable languages, e.g., $\overline{HALT}_{\text{TM}}$) to the given language.

Example. Prove the following language is unrecognizable.

$$L_2 = \{\langle M \rangle \mid M \text{ is a TM and } M \text{ halts on all inputs of length} \geq 2020\}$$

Solution. It is enough to show that $\overline{A_{TM}} \leq_m L_2$. Define $f(\langle T, w \rangle) =$ a machine M which performs the following on input x:

1. On input x, simulate T with w for $|x|$ steps.
2. If T accepts w (in $|x|$ steps or less), go into an infinite loop. Otherwise, accept x.

Observe

- If T does not accept w, then for every input x, M accepts x. Thus $\langle M \rangle \in L_2$.
- If T accepts w in k steps, then M will loop on every word longer than k characters. In particular, $\langle M \rangle \notin L_2$. □

C. Appendix: Complexity

C.1. List of NPC Problems

3SAT Given a Boolean CNF formula (an AND of ORs) on n variables in which each clause has at most 3 literals, determine whether there is an assignment of T or F values to the n variables that satisfies the formula.

Clique Determine if G has a clique [*] of size at least k.

IndepSet Determine if G has an independent set [†] of size $\geq k$.

VertexCover Determine if G has a vertex cover of size $\leq k$.

DominatingSet Determine if G has a dominating set[‡] of size k.

SetCover Given a collection $\mathcal{S}$ of subsets of $[m]$ and $k \in \mathbb{Z}^+$, determine if there are k sets $S_1, \ldots, S_k \in \mathcal{S}$ such that $S_1 \cup \cdots \cup S_k = [m]$.

(Dir)HamPath Determine if G contains a (directed) Hamiltonian path. [§]

(Dir)HamCycle Determine if G contains a (directed) Hamiltonian cycle.

SubsetSum Given an array of n elements $w_1, w_2, \ldots, w_n$ and a target weight T, determine whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i = T$.

Partition Given binary representation of n integers $a_1, \ldots, a_n$, is there a set $S \subseteq [n]$ such that $\sum_{i \in S} a_i = \sum_{j \in [n] \setminus S} a_j$?

HittingSet Given a set S, a collection C of subsets of S, and a number k, is there a subset $T \subseteq S$ of size at most k such that $T \cap C_i = \varnothing$ for all $C_i \in C$?

C.2. Prove NP-Completeness

Exercise. Prove the following problem is **NP**-hard: Given binary representation of n integers $a_1, \ldots, a_n$, is there a set $S \subseteq [n]$ such that $\sum_{i \in S} a_i = \sum_{j \in [n] \setminus S} a_j$?

Solution. This problem is known as **Partition**. We reduce the SubsetSum problem to Partition.

- Problem: SubsetSum
  - Input: A set S of integers and an integer t.
  - Output: Is there a non-empty subset whose sum is t?
- Reduction: Given an instance of SubsetSum $\langle S, t \rangle$, we define the instance $\langle S' \rangle$ of Partition where $S' = S \cup \{s - 2t\}$ where $s = \sum s_i$. The construction clearly takes polynomial time.

We claim that $\langle S, t \rangle \in$ **SubsetSum** $\iff \langle S' \rangle \in$ **Partition**.

- If there exists a set of numbers $T \subseteq S$ that sum to t, then the remaining numbers in S sum to $s - t$. Therefore, if we let $T' = T \cup \{s - 2t\}$ then $(T', \bar{T}')$ is a partition of S' into two such that each partition sums to $s - t$.
- If there exists a partition of S' where each subset has sum $(s + s - 2t)/2 = s - t$. One of these sets contains the number $s - 2t$. Removing this number, we get a set of numbers whose sum is t; all of them are in S.

[*] a set of vertices where every two vertices are adjacent

[†] a set of vertices such that no two of which are adjacent

[‡] A **dominating set** for a graph $G = (V, E)$ is a subset S of V such that every vertex not in S is adjacent to at least one member of S.

[§] a path (cycle) that visits every vertex in G exactly once

D. Appendix: MISC

D.1. Rice's Theorem

Let $\mathcal{S}$ be a property. The language S_{TM} is defined as $S_{\text{TM}} = \{\langle M, \rangle \mid L(M) \in \mathcal{S}\}$.

Theorem. (Rice) *If $\mathcal{S}$ is a non-trivial property of Turing-recognizable languages, then S_{TM} is undecidable.*

The following is a proof template for related problems:

Problem. Show the language P is undecidable.

Solution. Suppose that P is decidable. Then there exists a halting TM M_P that recognizes the descriptions of TMs that satisfy P. In other words, given a description $\langle M \rangle$ of a TM M, M_P accepts $\langle M \rangle$ if *INSERT PROPERTY DESCRIPTION HERE* and rejects otherwise. We define a compute function f from the inputs of A_{TM} to the inputs of P:

$$f : \langle M, w \rangle \mapsto \left\langle T_{M,w} \right\rangle .$$

SHOW COMPUTABILITY OF THIS FUNCTION. We now define the TM $T_{M,w}$: On input x,

- Run M on w.
- If M accepts on w, run M_P on x. If M_P accepts, then accept.
- Otherwise, reject.

Analysis:

- If M accepts w, since M_P is a decider for P, $T_{M,w} \in P$.
- If M does not accept w (either rejects or never halts), then $T_{M,w} = \varnothing \notin P$.

Since A_{TM} is undecidable, P must be undecidable as well. □