

Computational Complexity and Computability

0

1

1

0



The cover depicts a Turing machine. Turing machine is a simple but powerful computation model used to study the computability and complexity of problems. It is considered one of the foundational models of theoretical computer science.

Copyright © 2022 Kevin Gao

TERRA-INCOGNITA.DEV

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Contents

I	Automata and Languages	
1	Regular Language	7
1.1	Alphabet and Strings	7
1.2	Regular Language	7
2	Context-Free Language	9
3	Turing Machine	11
3.1	Turing Machine	11
3.2	Configuration of Turing Machine	12
3.3	Decidability and Recognizability	13
3.4	Some Classes of Languages	13
3.5	Difference Between FSA and TM	13
3.6	Multi-tape Turing Machines	13
3.7	Non-deterministic Turing Machines	14
3.8	Enumerator	14
3.9	Church-Turing Thesis	16

II

Decidability and Computability

4	Undecidable Language	19
4.1	Countability	19
4.2	Universal Turing Machine	20

Appendix

Axioms & Theorems	23
Basic Prerequisite Mathematics	27
Proof Templates	33
Index	43
Bibliography	45
Courses	45
Books	45
Journal Articles	46

Automata and Languages

1	Regular Language	7
1.1	Alphabet and Strings	
1.2	Regular Language	
2	Context-Free Language	9
3	Turing Machine	11
3.1	Turing Machine	
3.2	Configuration of Turing Machine	
3.3	Decidability and Recognizability	
3.4	Some Classes of Languages	
3.5	Difference Between FSA and TM	
3.6	Multi-tape Turing Machines	
3.7	Non-deterministic Turing Machines	
3.8	Enumerator	
3.9	Church-Turing Thesis	

Chapter 1 Regular Language

1.1 Alphabet and Strings

Σ denotes a finite alphabet of symbols. Σ^* denotes a set of all strings, including the empty string ϵ , consist of elements of Σ .

For string $x \in \Sigma^*$, $|x|$ is the length of x . A language is a subset of Σ^* .

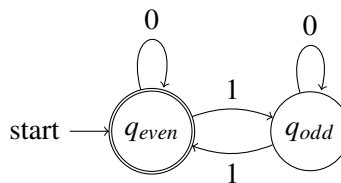
1.2 Regular Language

See 240 notes.

Example:

$\text{Even} = \{w \in \{0, 1\}^* \mid w \text{ contains an even number of 1's}\}$. It can be expressed using the regular expression $0^* + (0^*10^*10^*)^*$.

It also has a 2-state deterministic finite automaton that decides the language Even.



More formally,

Definition 1.2.1 — Deterministic Finite State Automata. A deterministic finite state automaton (DFA) is a quintuple (5-tuple)

$$M = (Q, \Sigma, \delta, s, F)$$

where

- Q is a finite set of states
- Σ is a finite set called the alphabet
- $\delta : Q \times \Sigma \rightarrow Q$, the transition function
- $s \in Q$, the starting state

- $F \subseteq Q$, the set of accepting states

Chapter 2 Context-Free Language

Chapter 3 Turing Machine

3.1 Turing Machine

While finite state automata and pushdown automata are all valid models of computation, they are too restricted as models of general purpose computers. Our goal is to have a model so that we can define computation as abstractly and general as possible.

How do we define algorithms rigorously? What does it mean for an algorithm to run in polynomial time. How do we argue that efficient algorithms do not exist.

Turing machine is a model of computation first proposed by Alan Turing in 1936. It is much more powerful than previous models that we have looked at. It is sufficiently general and can model what a human can do. A Turing machine can be think of a finite automata with unlimited and unrestricted memory.

In a Turing machine, we have an **one-way infinite tape** divided into “cells”, each holding one symbol, including the blank symbol \square . It also has a **read-write** head positioned in one square at a time that can move to the left or right. The control of the Turing is in of a fixed number of states. (The blank symbol \square is sometimes denoted by b or \square)

Initially, the input tape contains a finite number of symbols starting at the left-most cell with the remaining cells blank. The head starts at the left-most input symbol. Current state and symbol determine the next state, symbol written, and the movement of the head (either one square left or one square right).

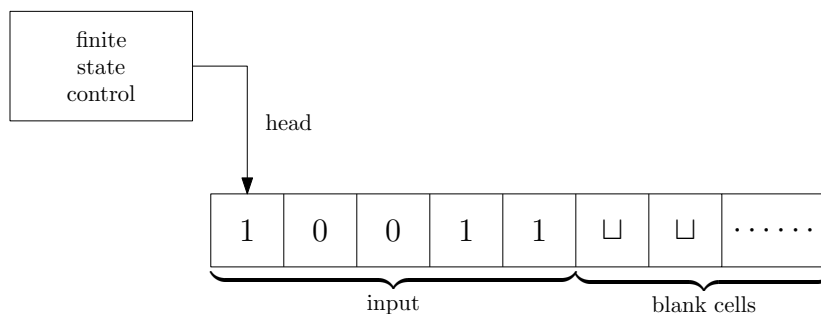


Figure 3.1: A Turing machine

Definition 3.1.1 — Turing Machine. A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q, Σ, Γ are all finite sets, and

- Q is the set of states
- Σ is the input alphabet not containing the blank symbol \sqcup
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$
- $\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state, where $q_{accept} \neq q_{reject}$

$\delta(q, a) = (q', a', x)$ where $x \in \{L, R\}$ means if a machine is in state q and head positioned on a square containing a , then the machine replaces a with a' , moves to state q' , and moves the head left (L) or right (R) depending on the direction given by x .

A Turing machine M works as follows on input string $x \in \Sigma^*$.

1. Initially $x = x_1x_2 \dots x_n \in \Sigma^*$ appears on leftmost n squares of the input tape. Rest of the tape is blank;
2. Head of M starts on the leftmost square of tape;
3. Initial state is q_0
4. M moves according to the transition function δ
5. Continue until M reaches q_{accept} or q_{reject} and then M halts. Otherwise, continue on forever.

Definition 3.1.2 — Language Recognized by Turing Machine. We say a Turing machine M **accepts** a string $x \in \Sigma^*$ if M upon reading the input x eventually **halts** in state q_{accept} .

Let $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$. We say $L(M)$ is the **language recognized/accepted by M** . We say $x \notin L(M)$ if M upon reading x either **halts** in q_{reject} or **loops**.

■ **Example 3.1 — Palindrome.**

$$\text{Palindrome} = \{yy^R \mid y \in \{0, 1\}^*\}$$

At a high level, the **Turing machine** that decides this language scans back and forth, matching and erasing leftmost and rightmost symbols. Accept if the input is completely erased or reject if otherwise.
■

3.2 Configuration of Turing Machine

The **configuration** of a Turing machine describes the state, head position, and tape contents. It is denoted xqy where $x, y \in \Gamma^*$ and $q \in Q$. In this notation, the head position is implied to be the leftmost symbol of y .

Note that xqy and $xqy\sqcup$ are equivalent configurations, but xqy and $\sqcup xqy$ are not equivalent. Whether or not the first symbol is blank is important.

Definition 3.2.1 Given two configurations C_1 and C_2 , we say configuration C_1 **yields** C_2 if C_2 follows from C_1 by one step of M (one application of δ).

■ **Example 3.2** Let $x, y \in \Gamma^*$ and $a, b \in \Gamma$. If $\delta(q, a) = (q', a', R)$, then $xqay$ yields $xa'q'y$. If $\delta(q, b) = (q', b', L)$, then $xqby$ yields $xq'ab'y$. qby yields $q'b'y$ because the tape head cannot move left anymore.

■

Definition 3.2.2 — Computation. The **computation** of M on input $x \in \Sigma^*$ is the sequence $C_0C_1C_2\ldots$ where $C_0 = q_0x$ and each configuration follows from the previous one. We say the computation is **halting** if it eventually reaches accept or reject state. Otherwise, we say the computation is **looping** (infinite).

3.3 Decidability and Recognizability

■ **Definition 3.3.1 — Decider.** A Turing machine M is a decider if it halts on all inputs $x \in \Sigma^*$.

■ **Definition 3.3.2 — Turing Decidable and Turing Recognizable.** A language $A \in \Sigma^*$ is Turing decidable if and only if there is a **decider** M such that $L(M) = A$.

A is semidecidable/Turing recognizable if and only if there is a Turing machine M such that $L(M) = A$. In this case, M may not halt if it does not accept A (reject by looping).

3.4 Some Classes of Languages

$D = \{A \subseteq \Sigma^* \mid A \text{ is decidable}\}$

$SD = \{A \subseteq \Sigma^* \mid A \text{ is semidecidable}\}$

$P = \{A \subseteq \Sigma^* \mid A = L(M) \text{ for some } M \text{ that halts in polynomial time}\}$

Later, we will show that $P \subsetneq D \subsetneq SD$.

3.5 Difference Between FSA and TM

- TM can read and write symbols. Infinite tape.
- Head can move left or right, unless the head is at left-most position.
- Special “accept” and “reject” states that stop computation immediately. The machine halts only when it reaches an accept or reject state. On the other hand, an FSA can only perform a finite amount of transitions before it halts.

3.6 Multi-tape Turing Machines

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

We will show a 2-tape TM can be simulated by a single tape TM. The idea is to store tape 1 on odd cells, tape 2 on even cells, and represent the tape head using new symbols with \cdot on top.

Theorem 3.6.1 Every multi-tape Turing machine is equivalent to an ordinary Turing machine.

To simulate a transition:

- scan the entire tape to find the symbols at tape heads;
- transition
- scan again and update the values at each tape and move tape heads left or right (by shifting the dots)

3.7 Non-deterministic Turing Machines

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

Instead of giving a single instruction, the transition function for TMs gives a set of instructions.

Theorem 3.7.1 Every non-deterministic Turing machine is equivalent to an ordinary Turing.

We can view the computation of a nondeterministic as a tree.

The idea is to run a BFS on the tree to find an accepting path. DFS may not work because it may get stuck in a looping state. At each state (internal node), the branching factor is at most b . We will address each node in level k the tree with a string of length k in $\{1, 2, \dots, b\}^k$. A BFS is then searching through $\{1, 2, \dots, b\}^k$ in standard string order (i.e. $\epsilon, 1, 2, \dots, b, 11, 12, \dots$).

We will simulate this with a 3-tape TM. The first tape stores the input. The second tape is for simulation. The third tape is the address incrementor.

NONDETERMINISTIC-TM

```

1  while true
2      run  $M$  on tape 2
3      whenever we need to make a choice, consult tape 3
4      if  $q_{accept}$  accept
5      increment tape 3

```

3.8 Enumerator

An enumerator E is a 2-tape Turing machine with a work tape and an output tape (printer). The output tape is readonly. More formally, an enumerator is a tuple $E = (Q, \Sigma, \Gamma, \delta, q_0, q_{print}, q_{reject})$ where

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \times \Sigma_\epsilon$$

is the transition function and Σ is the output alphabet. The purpose of the print state q_{print} is to end the printing of the current string (we can think of this as a carriage return on a typewriter). If q_{reject} is entered, the computation/printing will stop.

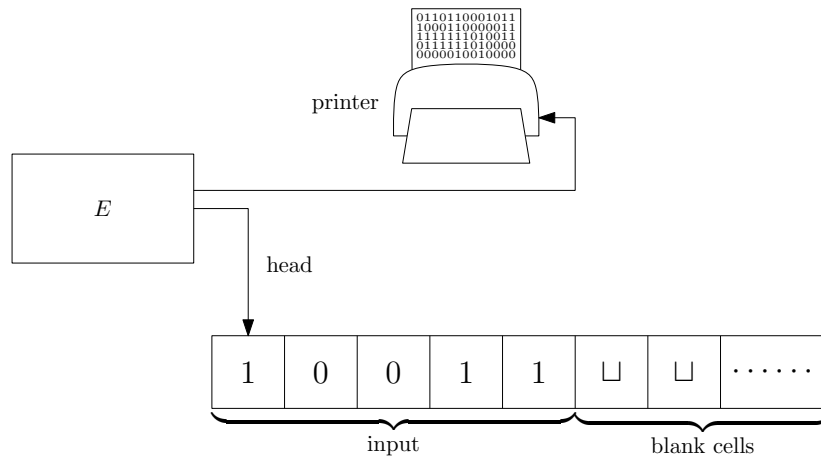


Figure 3.2: An enumerator.

Theorem 3.8.1 A language $A \subseteq \Sigma^*$ is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

(\Leftarrow): Suppose E enumerates a language A , we will show that there exists a TM M that recognizes A . We will construct a Turing machine M such that $L(M) = A$ as follows

M = “On input w ,

1. Run E . Every time E outputs a string, compare it with w ;
2. If w ever appears in the output E , accept; Otherwise, keep running”

M accepts those strings that are printed by E . Note that M does not necessarily terminate and hence $L(M)$ is not necessarily decidable.

(\Rightarrow): Suppose A is Turing-recognizable, then $L(M) = A$ for some Turing machine M . Let $s_1, s_2, s_3, \dots \in \Sigma^*$ be the list of all possible strings in Σ^* in lexicographical ordering, and construct E as follows:

- E = “1. Repeat the following for $i = 1, 2, 3, \dots$
2. Run M for i steps on each input s_1, s_2, \dots, s_i ;
 3. If any computations accept, print out the corresponding s_j ”

■

It might be tempting to run M on all possible strings at each iteration or to use a nondeterministic Turing machine, but both might get stuck running forever because there can be infinitely many possible strings to consider. In the case of a nondeterministic Turing machine, the computation tree can have a infinite branching factor.

Theorem 3.8.2 A language is decidable if and only if some enumerator E enumerates it in lexicographic order.

3.9 Church-Turing Thesis



Decidability and Computability

4	Undecidable Language	19
4.1	Countability	
4.2	Universal Turing Machine	

Chapter 4 Undecidable Language

4.1 Countability

Definition 4.1.1 — Countable Set. A set A is countable if A is empty or there is an injective function $f : A \rightarrow \mathbb{N}$, or equivalently, a surjective function $f : \mathbb{N} \rightarrow A$.

Theorem 4.1.1 \mathbb{R} is not countable.

Theorem 4.1.2 $2^{\mathbb{N}} = \mathcal{P}(\mathbb{N})$ is not countable.

Corollary 4.1.3 There exists some $A \subseteq \Sigma^*$ such that A is not Turing-recognizable.

Proof. We will prove this by showing that the set of all languages in Σ^* is not countable, but the set of Turing-recognizable languages is countable.

We first show that Σ^* is countable. To show this, we simply list all strings in Σ^* in lexicographic order.

Observe that $\text{TR} = \{A \subseteq \Sigma^* \mid A = L(M)\}$ is also countable because the number of Turing machines is countable, each of which can be encoded as a finite length string.

But the set of all languages $\{A \subseteq \Sigma^*\} = 2^{\Sigma^*}$. By Cantor's theorem, since Σ^* is countably infinite, the power set of Σ^* is uncountable.

Hence, there exists some language $A \subseteq \Sigma^*$ that is not Turing-recognizable. ■

Let $\langle M \rangle$ denote the encoding of a Turing machine M . For $n \in \mathbb{N}$, let $\langle n \rangle \in \{0, 1\}^*$ be the binary encoding of n . For a sequence of numbers n_1, n_2, \dots, n_k , $\langle n_1, n_2, \dots, n_k \rangle \in \{0, 1, \#\}^*$ be $\langle n_1 \rangle \# \langle n_2 \rangle \# \dots \# \langle n_k \rangle$.

Then, any Turing machine can be determined by a finite sequence of numbers specifying $|Q|$, $|\Sigma|$, $|\Gamma|$, identity of q_0 , q_{accept} , q_{reject} , and a sequence of numbers specifying δ .

$\langle M \rangle$ = the string that determines M

Furthermore, we can denote a Turing machine T with input w as $\langle M, w \rangle$.

Theorem 4.1.4 Let

$$\text{DIAG} = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin L(M) \}$$

DIAG is not Turing recognizable.

Proof. Suppose, for contradiction, that there exists a TM M such that $T(M) = \text{DIAG}$. Then, we can ask if $\langle M \rangle \in L(M)$.

Suppose that $\langle M \rangle \in L(M)$. Then

$$\langle M \rangle \in L(M) \text{ IFF } \langle M \rangle \in \text{DIAG} \text{ IFF } \langle M \rangle \notin L(M)$$

which is a contradiction. ■

This is similar to Russell's paradox.

Theorem 4.1.5 Every finite language is Turing recognizable and decidable.

Theorem 4.1.6 Every regular language is Turing recognizable and decidable.

4.2 Universal Turing Machine

Let A_{TM} be the language

$$\{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

Theorem 4.2.1 A_{TM} is Turing-recognizable.

Proof. We show that there exists a TM U such that $L(U) = A_{\text{TM}}$. U decodes $\langle M \rangle$ into a TM M and simulates M on w . Thus, A_{TM} is Turing-recognizable. ■

The U that we have constructed to recognize A_{TM} is called a universal Turing machine.

Theorem 4.2.2 A_{TM} is not decidable.

Commonly Used Axioms & Theorems

Rules of Inference

Axiom 1 — Modus Ponens. $(P \wedge (P \text{ IMPLIES } Q)) \text{ IMPLIES } Q$

Axiom 2 — Modus Tollens. $(\neg Q \wedge (P \text{ IMPLIES } Q)) \text{ IMPLIES } \neg P$

Axiom 3 — Hypothetical Syllogism (transitivity).

$((P \text{ IMPLIES } Q) \wedge (Q \text{ IMPLIES } R)) \text{ IMPLIES } (P \text{ IMPLIES } R)$

Axiom 4 — Disjunctive Syllogism. $((P \vee Q) \wedge \neg P) \text{ IMPLIES } Q$

Axiom 5 — Addition. $P \text{ IMPLIES } (P \vee Q)$

Axiom 6 — Simplification. $(P \wedge Q) \text{ IMPLIES } P$

Axiom 7 — Conjunction. $((P) \wedge (Q)) \text{ IMPLIES } (P \wedge Q)$

Axiom 8 — Resolution. $((P \vee Q) \wedge (\neg P \vee R)) \text{ IMPLIES } (Q \vee R)$

Laws of Logic

Axiom 9 — Implication Law. $(P \text{ IMPLIES } Q) \equiv (\neg P \vee Q)$

Axiom 10 — Distributive Law.

$$(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$$

$$(P \vee (Q \wedge R)) \equiv ((P \vee Q) \wedge (P \vee R))$$

Axiom 11 — De Morgan's Law.

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

Axiom 12 — Absorption Law.

$$(P \vee (P \wedge Q)) \equiv P$$

$$(P \wedge (P \vee Q)) \equiv P$$

Axiom 13 — Commutativity of AND. $A \wedge B \equiv B \wedge A$

Axiom 14 — Associativity of AND. $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$

Axiom 15 — Identity of AND. $\mathbf{T} \wedge A \equiv A$

Axiom 16 — Zero of AND. $\mathbf{F} \wedge A \equiv \mathbf{F}$

Axiom 17 — Idempotence for AND. $A \wedge A \equiv A$

Axiom 18 — Contradiction for AND. $A \wedge \neg A \equiv \mathbf{F}$

Axiom 19 — Double Negation. $\neg(\neg A) \equiv A$

Axiom 20 — Validity for OR. $A \vee \neg A \equiv \mathbf{T}$

Induction

Axiom 21 — Well Ordering Principle. Every nonempty set of nonnegative integers has a smallest element. i.e., For any $A \subset \mathbb{N}$ such that $A \neq \emptyset$, there is some $a \in A$ such that $\forall a' \in A. a \leq a'$.

Recurrences

Theorem 22 — The Master Theorem. Suppose that for $n \in \mathbb{Z}^+$.

$$T(n) = \begin{cases} c & \text{if } n < B \\ a_1 T(\lceil n/b \rceil) + a_2 T(\lfloor n/b \rfloor) + dn^i & \text{if } n \geq B \end{cases}$$

where $a_1, a_2, B, b \in \mathbb{N}$.

Let $a = a_1 + a_2 \geq 1$, $b > 1$, and $c, d, i \in \mathbb{R} \cup \{0\}$. Then,

$$T(n) \in \begin{cases} O(n^i \log n) & \text{if } a = b^i \\ O(n^i) & \text{if } a < b^i \\ O(n^{\log_b a}) & \text{if } a > b^i \end{cases}$$

Linear Recurrences:

A linear recurrence is an equation

$$f(n) = \underbrace{a_1 f(n-1) + a_2 f(n-2) + \cdots + a_d f(n-d)}_{\text{homogeneous part}} + \underbrace{g(n)}_{\text{inhomogeneous part}}$$

along with some boundary conditions.

The procedure for solving linear recurrences are as follows:

1. Find the roots of the characteristic equation. Linear recurrences usually have exponential solutions (such as x^n). Such solution is called the **homogeneous solution**.

$$x^n = a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{k-1} x + a_k$$

2. Write down the homogeneous solution. Each root generates one term and the homogeneous solution is their sum. A non-repeated root r generates the term cr^n , where c is a constant to be determined later. A root with r with multiplicity k generates the terms

$$d_1 r^n \quad d_2 n r^n \quad d_3 n^2 r^n \quad \cdots \quad d_k n^{k-1} r^n$$

where d_1, \dots, d_k are constants to be determined later.

3. Find a **particular solution** for the full recurrence including the inhomogeneous part, but without considering the boundary conditions.
If $g(n)$ is a constant or a polynomial, try a polynomial of the same degree, then of one higher degree, then two higher. If $g(n)$ is exponential in the form $g(n) = k^n$, then try $f(n) = ck^n$, then $f(n) = (bn + c)k^n$, then $f(n) = (an^2 + bn + c)k^n$, etc.
4. Write the **general solution**, which is the sum of homogeneous solution and particular solution.
5. Substitute the boundary condition into the general solution. Each boundary condition gives a linear equation. Solve such system of linear equations for the values of the constants to make the solution consistent with the boundary conditions.

Basic Prerequisite Mathematics

Basic Prerequisite Mathematics

SET THEORY

Common Sets

- $\mathbb{N} = \{0, 1, 2, \dots\}$: the natural numbers, or non-negative integers. The convention in computer science is to include 0 in the natural numbers.
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$: the integers
- $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$: the positive integers
- $\mathbb{Z}^- = \{-1, -2, -3, \dots\}$: the negative integers
- \mathbb{Q} the rational numbers, \mathbb{Q}^+ the positive rationals, and \mathbb{Q}^- the negative rationals.
- \mathbb{R} the real numbers, \mathbb{R}^+ the positive reals, and \mathbb{R}^- the negative reals.

Notation

For any sets A and B , we will use the following standard notation.

- $x \in A$: “ x is an element of A ” or “ A contains x ”
- $A \subseteq B$: “ A is a subset of B ” or “ A is included in B ”
- $A = B$: “ A equals B ” (Note that $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.)
- $A \subsetneq B$: “ A is a proper subset of B ”
(Note that $A \subsetneq B$ if and only if $A \subseteq B$ and $A \neq B$.)
- $A \cup B$: “ A union B ”
- $A \cap B$: “ A intersection B ”
- $A - B$: “ A minus B ” (*set* difference)
- $|A|$: “cardinality of A ” (the number of elements of A)
- \emptyset or $\{\}$: “the empty set”
- $\mathcal{P}(A)$ or 2^A : “powerset of A ” (the set of all subsets of A)
If $A = \{a, 34, \triangle\}$, then $\mathcal{P}(A) = \{\{\}, \{a\}, \{34\}, \{\triangle\}, \{a, 34\}, \{a, \triangle\}, \{34, \triangle\}, \{a, 34, \triangle\}\}$.
 $S \in \mathcal{P}(A)$ means the same as $S \subseteq A$.
- $\{x \in A \mid P(x)\}$: “the set of elements x in A for which $P(x)$ is true”
For example, $\{x \in \mathbb{Z} \mid \cos(\pi x) > 0\}$ represents the set of integers x for which $\cos(\pi x)$ is greater than zero, *i.e.*, it is equal to $\{\dots, -4, -2, 0, 2, 4, \dots\} = \{x \in \mathbb{Z} \mid x \text{ is even}\}$.

- $A \times B$: “the cross product or Cartesian product of A and B ”
 $A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$.
 If $A = \{1, 2, 3\}$ and $B = \{5, 6\}$, then $A \times B = \{(1, 5), (1, 6), (2, 5), (2, 6), (3, 5), (3, 6)\}$.
- A^n : “the cross product of n copies of A ”
 This is set of all sequences of $n \geq 1$ elements, each of which is in A .
- B^A or $A \rightarrow B$: “the set of all functions from A to B .”
- $f : A \rightarrow B$ or $f \in B^A$: “ f is a function from A to B ”
 f associates one element $f(x) \in B$ to every element $x \in A$.

NUMBER THEORY

For any two natural numbers a and b , we say that a *divides* b if there exists a natural number c such that $b = ac$. In such a case, we say that a is a *divisor* of b (*e.g.*, 3 is a divisor of 12 but 3 is not a divisor of 16). Note that any natural number is a divisor of 0 and 1 is a divisor of any natural number. A number a is *even* if 2 divides a and is *odd* if 2 does not divide a .

A natural number p is *prime* if it has exactly two positive divisors (*e.g.*, 2 is prime since its positive divisors are 1 and 2 but 1 is **not** prime since it only has one positive divisor: 1). There are an infinite number of prime numbers and any integer greater than one can be expressed in a unique way as a finite product of prime numbers (*e.g.*, $8 = 2^3$, $77 = 7 \times 11$, $3 = 3$).

Inequalities

For any integers m and n , $m < n$ if and only if $m + 1 \leq n$ and $m > n$ if and only if $m \geq n + 1$. For any real numbers w , x , y , and z , the following properties always hold (they also hold when $<$ and \leq are exchanged throughout with $>$ and \geq , respectively).

- if $x < y$ and $w \leq z$, then $x + w < y + z$
- if $x < y$, then
$$\begin{cases} xz < yz & \text{if } z > 0 \\ xz = yz & \text{if } z = 0 \\ xz > yz & \text{if } z < 0 \end{cases}$$
- if $x \leq y$ and $y < z$ (or if $x < y$ and $y \leq z$), then $x < z$

Functions

Here are some common number-theoretic functions together with their definitions and properties of them. (Unless noted otherwise, in this section, x and y represent arbitrary real numbers and k , m , and n represent arbitrary positive integers.)

- $\min\{x, y\}$: “minimum of x and y ” (the smallest of x or y)
 Properties: $\min\{x, y\} \leq x$
 $\min\{x, y\} \leq y$

- $\max\{x, y\}$: “maximum of x and y ” (the largest of x or y)
 Properties: $x \leq \max\{x, y\}$
 $y \leq \max\{x, y\}$
- $\lfloor x \rfloor$: “floor of x ” (the greatest integer less than or equal to x , *e.g.*, $\lfloor 5.67 \rfloor = 5$, $\lfloor -2.01 \rfloor = -3$)
 Properties: $x - 1 < \lfloor x \rfloor \leq x$
 $\lfloor -x \rfloor = -\lceil x \rceil$
 $\lfloor x + k \rfloor = \lfloor x \rfloor + k$
 $\lfloor \lfloor k/m \rfloor / n \rfloor = \lfloor k/mn \rfloor$
 $(k - m + 1)/m \leq \lfloor k/m \rfloor$
- $\lceil x \rceil$: “ceiling of x ” (the least integer greater than or equal to x , *e.g.*, $\lceil 5.67 \rceil = 6$, $\lceil -2.01 \rceil = -2$)
 Properties: $x \leq \lceil x \rceil < x + 1$
 $\lceil -x \rceil = -\lfloor x \rfloor$
 $\lceil x + k \rceil = \lceil x \rceil + k$
 $\lceil \lceil k/m \rceil / n \rceil = \lceil k/mn \rceil$
 $\lceil k/m \rceil \leq (k + m - 1)/m$
 Additional property of $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$: $\lfloor k/2 \rfloor + \lceil k/2 \rceil = k$.
- $|x|$: “absolute value of x ” ($|x| = x$ if $x \geq 0$; $-x$ if $x < 0$, *e.g.*, $|5.67| = 5.67$, $|-2.01| = 2.01$)
 BEWARE! The same notation is used to represent the cardinality $|A|$ of a set A and the absolute value $|x|$ of a number x so be sure you are aware of the context in which it is used.
- $m \operatorname{div} n$: “the quotient of m divided by n ” (integer division of m by n , *e.g.*, $5 \operatorname{div} 6 = 0$, $27 \operatorname{div} 4 = 6$, $-27 \operatorname{div} 4 = -6$)
 Properties: If $m, n > 0$, then $m \operatorname{div} n = \lfloor m/n \rfloor$
 $(-m) \operatorname{div} n = -(m \operatorname{div} n) = m \operatorname{div} (-n)$
- $m \operatorname{rem} n$: “the remainder of m divided by n ” (*e.g.*, $5 \operatorname{rem} 6 = 5$, $27 \operatorname{rem} 4 = 3$, $-27 \operatorname{rem} 4 = -3$)
 Properties: $m = (m \operatorname{div} n) \cdot n + m \operatorname{rem} n$
 $(-m) \operatorname{rem} n = -(m \operatorname{rem} n) = m \operatorname{rem} (-n)$
- $m \bmod n$: “ m modulo n ” (*e.g.*, $5 \bmod 6 = 5$, $27 \bmod 4 = 3$, $-27 \bmod 4 = 1$)
 Properties: $0 \leq m \bmod n < n$
 n divides $m - (m \bmod n)$.
- $\gcd(m, n)$: “greatest common divisor of m and n ” (the largest positive integer that divides both m and n)
 For example, $\gcd(3, 4) = 1$, $\gcd(12, 20) = 4$, $\gcd(3, 6) = 3$
- $\operatorname{lcm}(m, n)$: “least common multiple of m and n ” (the smallest positive integer that m and n both divide)
 For example, $\operatorname{lcm}(3, 4) = 12$, $\operatorname{lcm}(12, 20) = 60$, $\operatorname{lcm}(3, 6) = 6$
 Properties: $\gcd(m, n) \cdot \operatorname{lcm}(m, n) = m \cdot n$.

CALCULUS

Limits and Sums

An infinite sequence of real numbers $\{a_n\} = a_1, a_2, \dots, a_n, \dots$ *converges* to a limit $L \in \mathbb{R}$ if, for every $\varepsilon > 0$, there exists $n_0 \geq 0$ such that $|a_n - L| < \varepsilon$ for every $n \geq n_0$. In this case, we write $\lim_{n \rightarrow \infty} a_n = L$. Otherwise, we say that the sequence *diverges*.

If $\{a_n\}$ and $\{b_n\}$ are two sequences of real numbers such that $\lim_{n \rightarrow \infty} a_n = L_1$ and $\lim_{n \rightarrow \infty} b_n = L_2$, then

$$\lim_{n \rightarrow \infty} (a_n + b_n) = L_1 + L_2 \quad \text{and} \quad \lim_{n \rightarrow \infty} (a_n \cdot b_n) = L_1 \cdot L_2.$$

In particular, if c is any real number, then

$$\lim_{n \rightarrow \infty} (c \cdot a_n) = c \cdot L_1.$$

The sum $a_1 + a_2 + \dots + a_n$ and product $a_1 \cdot a_2 \cdot \dots \cdot a_n$ of the finite sequence a_1, a_2, \dots, a_n are denoted by

$$\sum_{i=1}^n a_i \quad \text{and} \quad \prod_{i=1}^n a_i.$$

If the elements of the sequence are all different and $S = \{a_1, a_2, \dots, a_n\}$ is the set of elements in the sequence, these can also be denoted by

$$\sum_{a \in S} a \quad \text{and} \quad \prod_{a \in S} a.$$

Examples:

- For any $a \in \mathbb{R}$ such that $-1 < a < 1$, $\lim_{n \rightarrow \infty} a^n = 0$.
- For any $a \in \mathbb{R}^+$, $\lim_{n \rightarrow \infty} a^{1/n} = 1$.
- For any $a \in \mathbb{R}^+$, $\lim_{n \rightarrow \infty} (1/n)^a = 0$.
- $\lim_{n \rightarrow \infty} (1 + 1/n)^n = e = 2.71828182845904523536 \dots$

- For any $a, b \in \mathbb{R}$, the *arithmetic* sum is given by:

$$\sum_{i=0}^n (a + ib) = (a) + (a + b) + (a + 2b) + \dots + (a + nb) = \frac{1}{2}(n+1)(2a + nb).$$

- For any $a, b \in \mathbb{R}^+$, the *geometric* sum is given by:

$$\sum_{i=0}^n (ab^i) = a + ab + ab^2 + \dots + ab^n = \frac{a(1 - b^{n+1})}{1 - b}.$$

EXPONENTS AND LOGARITHMS

Definition: For any $a, b, c \in \mathbb{R}^+$, $a = \log_b c$ if and only if $b^a = c$.

Notation: For any $x \in \mathbb{R}^+$, $\ln x = \log_e x$ and $\lg x = \log_2 x$.

For any $a, b, c \in \mathbb{R}^+$ and any $n \in \mathbb{Z}^+$, the following properties always hold.

- $\sqrt[n]{b} = b^{1/n}$
- $b^a b^c = b^{a+c}$
- $(b^a)^c = b^{ac}$
- $b^a / b^c = b^{a-c}$
- $b^0 = 1$
- $a^b c^b = (ac)^b$
- $b^{\log_b a} = a = \log_b b^a$
- $a^{\log_b c} = c^{\log_b a}$
- $\log_b(ac) = \log_b a + \log_b c$
- $\log_b(a^c) = c \cdot \log_b a$
- $\log_b(a/c) = \log_b a - \log_b c$
- $\log_b 1 = 0$
- $\log_b a = \log_c a / \log_c b$

BINARY NOTATION

A *binary number* is a sequence of bits $a_k \cdots a_1 a_0$ where each bit a_i is equal to 0 or 1. Every binary number represents a natural number in the following way:

$$(a_k \cdots a_1 a_0)_2 = \sum_{i=0}^k a_i 2^i = a_k 2^k + \cdots + a_1 2 + a_0.$$

For example, $(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 1 = 9$, $(01110)_2 = 8 + 4 + 2 = 14$.

Properties:

- If $a = (a_k \cdots a_1 a_0)_2$, then $2a = (a_k \cdots a_1 a_0 0)_2$, *e.g.*, $9 = (1001)_2$ so $18 = (10010)_2$.
- If $a = (a_k \cdots a_1 a_0)_2$, then $\lfloor a/2 \rfloor = (a_k \cdots a_1)_2$, *e.g.*, $9 = (1001)_2$ so $4 = (100)_2$.
- The smallest number of bits required to represent the positive integer n in binary is called the *length* of n and is equal to $\lceil \log_2(n+1) \rceil$.

Make sure you know how to add and multiply two binary numbers. For example, $(1111)_2 + (101)_2 = (10100)_2$ and $(1111)_2 \times (101)_2 = (1001011)_2$.

Proof Templates

Proof Outlines

LINE NUMBERS: Only lines that are referred to have labels (for example, L1) in this document. For a formal proof, all lines are numbered. Line numbers appear at the beginning of a line. You can indent line numbers together with the lines they are numbering or all line numbers can be unindented, provided you are consistent.

INDENTATION: Indent when you make an assumption or define a variable. Unindent when this assumption or variable is no longer being used.

1. **Implication:** Direct proof of $A \text{ IMPLIES } B$.

L1. Assume A .
:
L2. B
 $A \text{ IMPLIES } B$; direct proof: L1, L2

2. **Implication:** Indirect proof of $A \text{ IMPLIES } B$.

L1. Assume $\text{NOT}(B)$.
:
L2. $\text{NOT}(A)$
 $A \text{ IMPLIES } B$; indirect proof: L1, L2

3. **Equivalence:** Proof of $A \text{ IFF } B$.

L1. Assume A .
:
L2. B
L3. $A \text{ IMPLIES } B$; direct proof: L1, L2
L4. Assume B .
:
L5. A
L6. $B \text{ IMPLIES } A$; direct proof: L4, L5
 $A \text{ IFF } B$; equivalence: L3, L6

4. **Proof by contradiction** of A .

L1. To obtain a contradiction, assume $\text{NOT}(A)$.
:
L2. B
:
L3. $\text{NOT}(B)$
L4. This is a contradiction: L2, L3
Therefore A ; proof by contradiction: L1, L4

5. **Modus Ponens.**

⋮
L1. A
⋮
L2. $A \text{ IMPLIES } B$
 B ; modus ponens: L1, L2

6. **Conjunction:** Proof of $A \text{ AND } B$:

⋮
L1. A
⋮
L2. B
 $A \text{ AND } B$; proof of conjunction; L1, 2

7. **Use of Conjunction:**

⋮
L1. $A \text{ AND } B$
 A ; use of conjunction: L1
 B ; use of conjunction: L1

8. **Implication with Conjunction:** Proof of $(A_1 \text{ AND } A_2) \text{ IMPLIES } B$.

L1. Assume $A_1 \text{ AND } A_2$.
 A_1 ; use of conjunction, L1
 A_2 ; use of conjunction, L1
⋮
L2. B
 $(A_1 \text{ AND } A_2) \text{ IMPLIES } B$; direct proof, L1, L2

9. **Implication with Conjunction:** Proof of $A \text{ IMPLIES } (B_1 \text{ AND } B_2)$.

L1. Assume A .
⋮
L2. B_1
⋮
L3. B_2
L4. $B_1 \text{ AND } B_2$; proof of conjunction: L2, L3
 $A \text{ IMPLIES } (B_1 \text{ AND } B_2)$; direct proof: L1, L4

10. **Disjunction:** Proof of $A \text{ OR } B$ and $B \text{ OR } A$.

⋮
L1. A
 $A \text{ OR } B$; proof of disjunction: L1
 $B \text{ OR } A$; proof of disjunction: L1

11. **Proof by cases.**

L1. C OR $NOT(C)$ tautology
L2. Case 1: Assume C .
 \vdots
 L3. A
L4. C IMPLIES A ; direct proof: L2, L3
L5. Case 2: Assume $NOT(C)$.
 \vdots
 L6. A
L7. $NOT(C)$ IMPLIES A ; direct proof: L5, L6
 A proof by cases: L1, L4, L7

12. **Proof by cases** of A OR B .

L1. C OR $NOT(C)$ tautology
L2. Case 1: Assume C .
 \vdots
 L3. A
 L4. A OR B ; proof of disjunction, L3
L5. C IMPLIES $(A$ OR $B)$; direct proof, L2, L4
L6. Case 2: Assume $NOT(C)$.
 \vdots
 L7. B
 L8. A OR B ; proof of disjunction, L7
L9. $NOT(C)$ IMPLIES $(A$ OR $B)$; direct proof: L6, L8
 A OR B ; proof by cases: L1, L5, L9

13. **Implication with Disjunction:** Proof by cases of $(A_1$ OR $A_2)$ IMPLIES B .

L1. Case 1: Assume A_1 .
 \vdots
 L2. B
L3. A_1 IMPLIES B ; direct proof: L1, L2
L4. Case 2: Assume A_2 .
 \vdots
 L5. B
L6. A_2 IMPLIES B ; direct proof: L4, L5
 $(A_1$ OR $A_2)$ IMPLIES B ; proof by cases: L3, L6

14. **Implication with Disjunction:** Proof by cases of $A \text{ IMPLIES } (B_1 \text{ OR } B_2)$.

L1. Assume A .
 L2. $C \text{ OR } \text{NOT}(C)$ tautology
 L3. Case 1: Assume C .
 \vdots
 L4. B_1
 L5. $B_1 \text{ OR } B_2$; disjunction: L4
 L6. $C \text{ IMPLIES } (B_1 \text{ OR } B_2)$; direct proof: L3, L5
 L7. Case 2: Assume $\text{NOT}(C)$.
 \vdots
 L8. B_2
 L9. $B_1 \text{ OR } B_2$; disjunction: L8
 L10. $\text{NOT}(C) \text{ IMPLIES } (B_1 \text{ OR } B_2)$; direct proof: L7, L9
 L11. $B_1 \text{ OR } B_2$; proof by cases: L2, L6, L10
 $A \text{ IMPLIES } (B_1 \text{ OR } B_2)$; direct proof. L1, L11

15. **Substitution of a Variable in a Tautology:**

Suppose P is a propositional variable, Q is a formula, and R' is obtained from R by replacing *every* occurrence of P by (Q) .

L1. R tautology
 R' ; substitution of all P by Q : L1

16. **Substitution of a Formula by a Logically Equivalent Formula:**

Suppose S is a subformula of R and R' is obtained from R by replacing *some* occurrence of S by S' .

L1. R
 L2. $S \text{ IFF } S'$
 L3. R' ; substitution of an occurrence of S by S' : L1, L2

17. **Specialization:**

L1. $c \in D$
 L2. $\forall x \in D. P(x)$
 $P(c)$; specialization: L1, L2

18. **Generalization:** Proof of $\forall x \in D. P(x)$.

L1. Let x be an arbitrary element of D .
 \vdots
 L2. $P(x)$
 Since x is an arbitrary element of D ,
 $\forall x \in D. P(x)$; generalization: L1, L2

19. **Universal Quantification with Implication:** Proof of $\forall x \in D.(P(x) \text{ IMPLIES } Q(x))$.

L1. Let x be an arbitrary element of D .

L2. Assume $P(x)$

\vdots

L3. $Q(x)$

L4. $P(x) \text{ IMPLIES } Q(x)$; direct proof: L2, L3

Since x is an arbitrary element of D ,

$\forall x \in D.(P(x) \text{ IMPLIES } Q(x))$; generalization: L1, L4

20. **Implication with Universal Quantification:** Proof of $(\forall x \in D.P(x)) \text{ IMPLIES } A$.

L1. Assume $\forall x \in D.P(x)$.

\vdots

L2. $a \in D$

$P(a)$; specialization: L1, L2

\vdots

L3. A

Therefore $(\forall x \in D.P(x)) \text{ IMPLIES } A$; direct proof: L1, L3

21. **Implication with Universal Quantification:** Proof of $A \text{ IMPLIES } (\forall x \in D.P(x))$.

L1. Assume A .

L2. Let x be an arbitrary element of D .

\vdots

L3. $P(x)$

Since x is an arbitrary element of D ,

L4. $\forall x \in D.P(x)$; generalization, L2, L3

$A \text{ IMPLIES } (\forall x \in D.P(x))$; direct proof: L1, L4

22. **Instantiation:**

L1. $\exists x \in D.P(x)$

Let $c \in D$ be such that $P(c)$; instantiation: L1

\vdots

23. **Construction:** Proof of $\exists x \in D.P(x)$.

L1. Let $a = \dots$

\vdots

L2. $a \in D$

\vdots

L3. $P(a)$

$\exists x \in D.P(x)$; construction: L1, L2, L3

24. **Existential Quantification with Implication:** Proof of $\exists x \in D.(P(x) \text{ IMPLIES } Q(x))$.

L1. Let $a = \dots$
 \vdots
L2. $a \in D$
L3. Suppose $P(a)$.
 \vdots
L4. $Q(a)$
L5. $P(a) \text{ IMPLIES } Q(a)$; direct proof: L3, L4
 $\exists x \in D.(P(x) \text{ IMPLIES } Q(x))$; construction: L1, L2, L5

25. **Implication with Existential Quantification:** Proof of $(\exists x \in D.P(x)) \text{ IMPLIES } A$.

L1. Assume $\exists x \in D.P(x)$.
Let $a \in D$ be such that $P(a)$; instantiation: L1
 \vdots
L2. A
 $(\exists x \in D.P(x)) \text{ IMPLIES } A$; direct proof: L1, L2

26. **Implication with Existential Quantification:** Proof of $A \text{ IMPLIES } (\exists x \in D.P(x))$.

L1. Assume A .
L2. Let $a = \dots$
 \vdots
L3. $a \in D$
 \vdots
L4. $P(a)$
L5. $\exists x \in D.P(x)$; construction: L2, L3, L4
 $A \text{ IMPLIES } (\exists x \in D.P(x))$; direct proof: L1, L5

27. **Subset:** Proof of $A \subseteq B$.

L1. Let $x \in A$ be arbitrary.
 \vdots
L2. $x \in B$
The following line is optional:
L3. $x \in A \text{ IMPLIES } x \in B$; direct proof: L1, L2
 $A \subseteq B$; definition of subset: L3 (or L1, L2, if the optional line is missing)

28. **Weak Induction:** Proof of $\forall n \in N. P(n)$

Base Case:

\vdots

L1. $P(0)$

L2. Let $n \in N$ be arbitrary.

L3. Assume $P(n)$.

\vdots

L4. $P(n+1)$

The following two lines are optional:

L5. $P(n)$ IMPLIES $(P(n+1))$; direct proof of implication: L3, L4

L6. $\forall n \in N. (P(n) \text{ IMPLIES } P(n+1))$; generalization L2, L5

$\forall n \in N. P(n)$ induction; L1, L6 (or L1, L2, L3, L4, if the optional lines are missing)

29. **Strong Induction:** Proof of $\forall n \in N. P(n)$

L1. Let $n \in N$ be arbitrary.

L2. Assume $\forall j \in N. (j < n \text{ IMPLIES } P(j))$

\vdots

L3. $P(n)$

The following two lines are optional:

L4. $\forall j \in N. (j < n \text{ IMPLIES } P(j)) \text{ IMPLIES } P(n)$; direct proof of implication: L2, L3

L5. $\forall n \in N. [\forall j \in N. (j < n \text{ IMPLIES } P(j)) \text{ IMPLIES } P(n)]$; generalization: L1, L4

$\forall n \in N. P(n)$; strong induction: L5 (or L1, L2, L3, if the optional lines are missing)

30. **Structural Induction:** Proof of $\forall e \in S. P(e)$, where S is a recursively defined set

Base case(s):

L1. For each base case e in the definition of S

L2. $P(e)$.

Constructor case(s):

L3. For each constructor case e of the definition of S ,

L4. assume $P(e')$ for all components e' of e .

\vdots

L5. $P(e)$

$\forall e \in S. P(e)$; structural induction: L1, L2, L3, L4, L5

31. **Well Ordering Principle:** Proof of $\forall e \in S. P(e)$, where S is a well ordered set, i.e. every nonempty subset of S has a smallest element.

L1. To obtain a contradiction, suppose that $\forall e \in S. P(e)$ is false.

L2. Let $C = \{e \in S \mid P(e) \text{ is false}\}$ be the set of counterexamples to P .

L3. $C \neq \emptyset$; definition: L1, L2

L4. Let e be the smallest element of C ; well ordering principle: L2, L3

Let $e' = \dots$

\vdots

L5. $e' \in C$

\vdots

L6. $e' < e$.

L7. This is a contradiction: L4, L5, L6

$\forall e \in S. P(e)$; proof by contradiction: L1, L7

Index

Bibliography

Courses

- [1] Allan Borodin. *CSC364*. University of Toronto. 2004.
- [2] Allan Borodin. *CSC373 Spring 2011*. University of Toronto. 2011.
- [3] Stephen A. Cook. *CSC463 Computational Complexity and Computability Winter 2018*. University of Toronto. 2018.
- [5] Faith Ellen. *CSC240 Enriched Introduction to Theory of Computation Winter 2021*. University of Toronto. 2021.
- [6] Faith Ellen. *CSC265 Enriched Data Structures and Analysis Fall 2021*. University of Toronto. 2021.
- [8] Shubhangi Saraf. *CSC463 Computational Complexity and Computability Winter 2022*. University of Toronto. 2022.

Books

- [4] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.
- [7] Jon Kleinberg and Éva Tardos. *Algorithm Design*. 1st. Pearson, 2005. ISBN: 978-0321295354.
- [9] Michael Sipser. *Introduction to the Theory of Computation*. Third. Boston, MA: Course Technology, 2013. ISBN: 113318779X.

Journal Articles