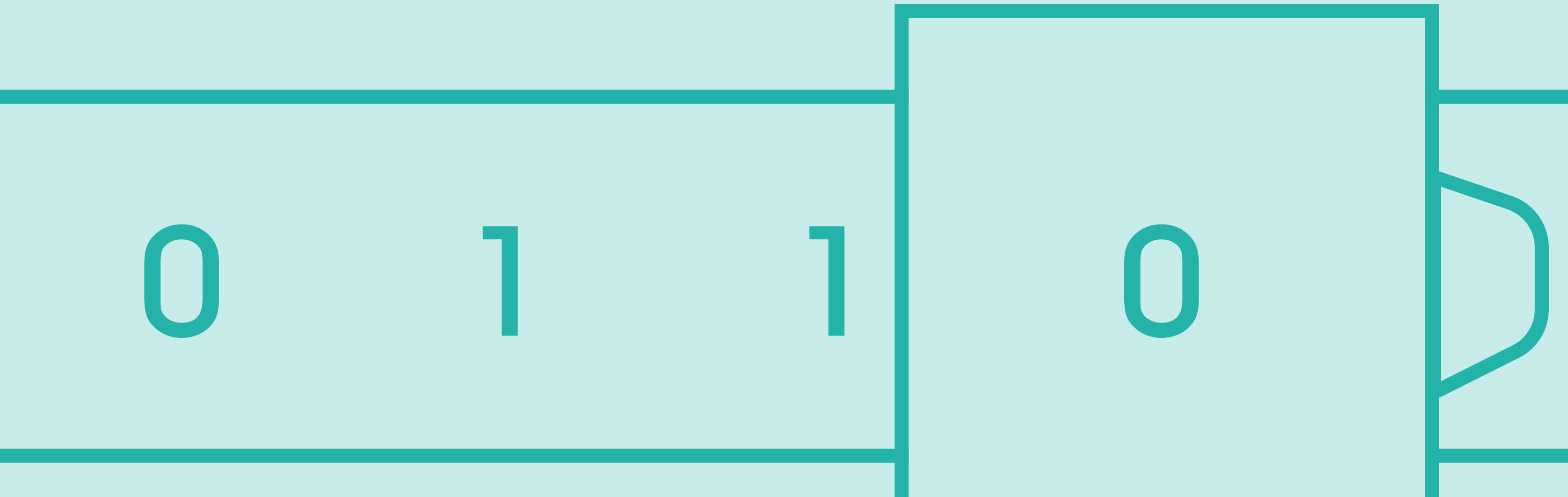
Computational Complexity and Computability



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

II Decidability and Computability

Appendix

Automata and Languages

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.

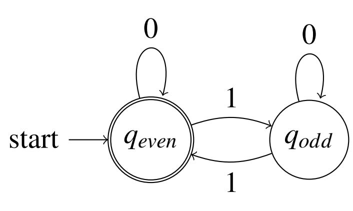
For string $x \in \Sigma^*$, $|x|$ is the length of x. A language is a subset of Σ^*.

1.2 Regular Language

See 240 notes.

Example:
$mathrmEven = \{w \in \{0,1\}^* \mid w$ 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
- $\delta : Q \times \Sigma \to Q$, 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 $\sqcup$. 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 $\sqcup$ is sometimes denoted by $\not b$ 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).

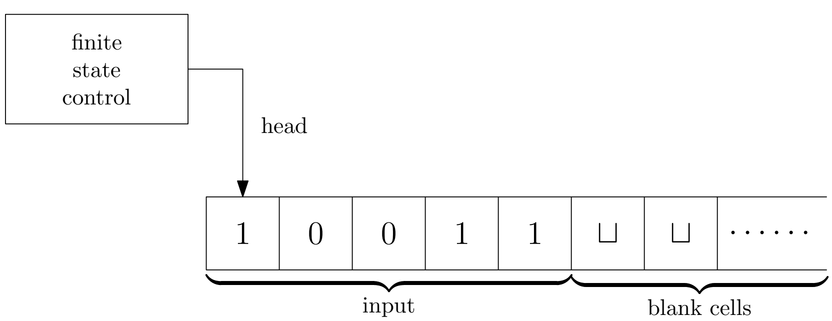


Figure 3.1: A Turing machine

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 $\Sigma \subset \Gamma$
- $\delta : (Q - \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ 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 $x = x_1 x_2 \dots x_n \in \Sigma^*$ 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 earased 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 $xaqby$ 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

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

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

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

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

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

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

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

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

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 cam 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 $\{1,2,\ldots,b\}^k$. A BFS is then searching through $\{1,2,\ldots,b\}^k$ in standard string order (i.e. $\varepsilon, 1, 2, \ldots, b, 11, 12, \ldots$).

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

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

$$\delta : \; Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \times \Sigma_\varepsilon$$

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.

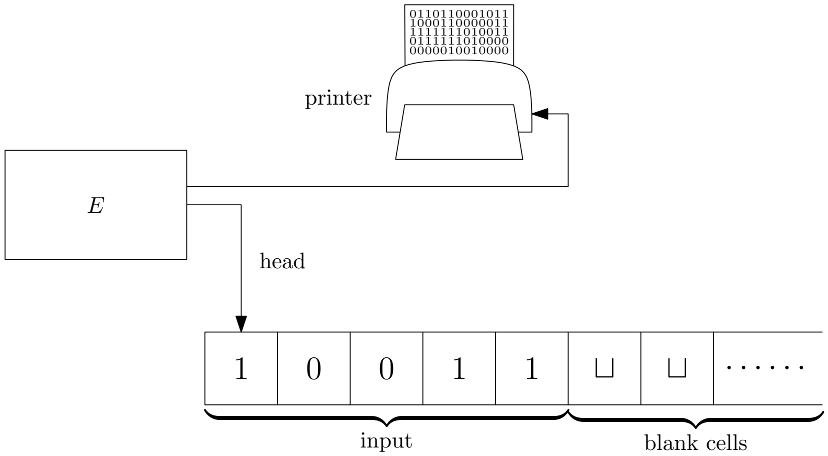


Figure 3.2: An enumerator.

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 $s_1, s_2, s_3, \ldots \in \Sigma^*$ be the list of all possible strings in Σ^* in lexicographical ordering, and construct E as follows:

$E =$ "1. Repeat the following for $i = 1, 2, 3, \ldots$

2. Run M for i steps on each input $s_1, s_2, \ldots, s_i$;
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

II

Decidability and Computability

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 a injective function $f : A \to \mathbb{N}$, or equivalently, a surjective function $f : \mathbb{N} \to A$.

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 $\mathsf{TR} = \{A \subseteq \Sigma^* \mid A = L(M)\}$ 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 $n_1, n_2, \ldots, n_k$, $\langle n_1, n_2, \ldots, n_k \rangle \in \{0,1,\#\}^*$ be $\langle n_1 \rangle \# \langle n_2 \rangle \# \cdots \# \langle n_k \rangle$.

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.

$$\langle M \rangle = \text{the string that determines } M$$

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

Theorem 4.1.4 Let

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

DIAG is not Turing recognizable.

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

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

$$\langle M \rangle \in L(M) \iff \langle M \rangle \in \mathsf{DIAG} \iff \langle M \rangle \notin\in L(M)$$

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 $L(U) = A_{\mathsf{TM}}$. 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.

4.3 Decidability

Theorem 4.3.1 A language L is decidable if and only if there is an enumerator that enumerates L in standard string order.

Proof. Decidable implies standard string order enumerator.

Let M be a decider for L. We will design an standard string order.

```
E
for w in standard string order
    run M on w
    if M accepts w
        print(w)
```

Standard string enumerator implies decidable.

```
M(w)
run E
while Has-Next(E)                 // while E still has string to print
    x = Next(E)                   // get next string
    if x == w
        accept
    if x ≥ w
        reject
```

■

Chapter 5 Reducibility

5.1 Proving Undecidability by Reduction

Theorem 5.1.1 A_{TF} is not decidable.

Proof. By contradiction.

Suppose that A_{TM} is decidable. Then there exists some Turing machine that decides A_{TM}. We will derive a contradiction by showing that if A_{TM} is decidable, then DIAG must also be decidable.

To decide $\langle M \rangle \in \mathsf{DIAG}$, we feed $\langle M, \langle M \rangle \rangle$ into a Turing machine that decides A_{TM}. ■

Corollary 5.1.2 Let D be the class of decidable languages, and let TR is the class of Turing-recognizable.

$$\mathsf{D} \subsetneq \mathsf{TR}$$

Let $\overline{A} = \{w \in \Sigma^* \mid w \notin A\} = \Sigma^* - A$.

Let $\text{co-}\mathsf{TR} = \{A \mid \overline{A} \in \mathsf{TR}\}$.

Theorem 5.1.3 If $A \subseteq \Sigma^*$, then $A \in \mathsf{D} \iff A \in \mathsf{TR} \wedge \overline{A} \in \mathsf{TR}$.

Proof. Suppose $A \in \mathsf{TR} \cap \text{co-}\mathsf{TR}$. Let M_1 be a TM that recognizes A. Let M_2 be a TM that recognizes $\overline{A}$.

Let M be a multi-tape Turing machine that simulates M_1 and M_2 in parallel. On input w, since $w \in A$ or $w \notin A$, either $M_1(w)$ accepts or $M_2(w)$ accepts. If M_1 accepts, then M accepts. If M_2 accepts, then M rejects. ■

5.2 Closure Properties

5.2.1 Union, Intersection, and Complement

Theorem 5.2.1 — Closure Under Union, Intersection, and Complement. If $A, B \in \mathsf{D}$, then $A \cup B \in \mathsf{D}$, $A \cap B \in \mathsf{D}$, and $\overline{A} \in \mathsf{D}$.

Theorem 5.2.2 The class of languages TR is closed under union, intersection, but NOT under complement.

An intuitive explanation for why TR is not closed under complement, then this would imply that all languages in TR would also be decidable.

Proof. Consider $A_{\mathsf{TM}} \in \mathsf{TR}$. If $\overline{A_{\mathsf{TM}}} \in \mathsf{TR}$, then $A_{\mathsf{TM}} \in \mathsf{co\text{-}TR}$, which implies that $A_{\mathsf{TM}} \in \mathsf{D}$. A contradiction. ■

Proof. Let $A, B \in \mathsf{TR}$. Then $A = L(M_1)$ and $B = L(M_2)$ for some TM M_1 and M_2.

$A \cap B = L(M_{A\cap B})$ where M_{AB} on input w runs M_1 and M_2 in parallel on two tapes and accepts if and only if M_1 and M_2 both halts and accepts.

$A \cup B = L(M_{A\cup B})$ where M_{AB} on input w runs M_1 and M_2 in parallel on two tapes and accepts if either M_1 or M_2 halts and accepts. ■

5.3 Computable Functions

So far, we have been using TM to accept and reject inputs. However, we can also use a TM to compute a function. If a function can be computed (evaluated) using a Turing machine, we say the function is computable.

Definition 5.3.1 — Computable Function. A function $f : \Sigma^* \to \Sigma^*$ is computable if there is some Turing machine M on every input $w \in \Sigma^*$, M halts with $f(w)$ on the tape.

■ **Example 5.1 — Duplicate.** Consider the function $f(w) = w \cdot w$ that duplicates a string w. ■

Note that f can take as input $\langle()M\rangle$ and output an encoding for some other Turing machine $\langle M'\rangle$.

■ **Example 5.2 — Example of Uncomputable Function - Halting Problem.** The function

$$h(\langle\langle M\rangle, w\rangle) = \begin{cases} 1 & \text{if } M \text{ halts on input } w \\ 0 & \text{if } M \text{ does not halt on input } w \end{cases}$$

solves the Halting problem and hence is not computable. ■

Theorem 5.3.1 — Church-Turing Thesis. Any function that can be computed by an "algorithm", then the function is computable.

It can be shown by the diagonal argument that the set of all functions is uncountable, and hence not all functions are computable because the set of computable functions is countable.

5.4 Mapping Reducible

Definition 5.4.1 — Mapping Reducible. A language A is mapping reducible to B, denoted $A \leq_m B$ if there is a computable function $f: \Sigma^* \to \Sigma^*$ where for every w,

$$w \in A \iff f(w) \in B$$

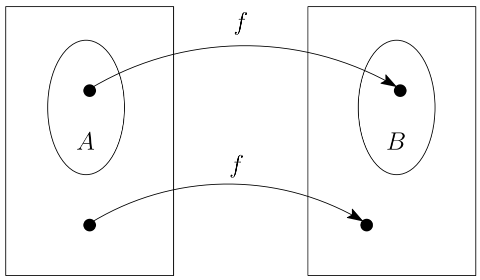


Figure 5.1: Function f reducing A to B.

Observation: If $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$. If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$ (this follows from function composition).

R Later, we will see other types of reductions. For example, $\leq_p$ for polytime reduction.

Theorem 5.4.1 If $A \leq_m B$ and B is decidable, then A is decidable. If B is Turing-recognizable, then A is also Turing-recognizable.

Proof. Let M be a decider for B. Let f be the computable mapping reduction from A to B, and let N be the Turing machine that computes this function.

Then A is decided by a Turing machine which on input w, runs N to obtain $f(w)$ and then runs M on $f(w)$. ■

Corollary 5.4.2 If $A \leq_m B$ and A is not decidable, then B is not decidable.

Application: Consider $\text{DIAG} = \{\langle M \rangle \mid \langle M \rangle \notin \mathcal{L}(M)\}$. Then, $\text{DIAG} \leq_m \overline{A_{\mathsf{TM}}}$ via the reduction $f: \Sigma^* \to \Sigma^*$ defined as

$$f(\langle M \rangle) = \langle M, \langle M \rangle \rangle$$

and if the input w to f is not an encoding of a TM, then

$$f(w) = \langle M_{accept}, \varepsilon \rangle$$

Since DIAG is not decidable, $\overline{A_{\mathsf{TM}}}$ is not decidable.

5.5 Applications of Reduction

5.5.1 Halting Problem

Consider the language

$$\text{HALT}_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ halts on } w\}$$

We have $\text{HALT}_{\mathsf{TM}} \leq_m A_{\mathsf{TM}}$ and $A_{\mathsf{TM}} \leq_m \text{HALT}_{\mathsf{TM}}$.

Proof.

($\text{HALT}_{\mathsf{TM}} \leq_m A_{\mathsf{TM}}$): If x is any string of the form $\langle M, w\rangle$, then $f(x) = \langle M', w\rangle$ where M' accepts if and only if M halts on input w. M' will simulate M running on w and accepts when M halts. If x is not of the correct form, $f(x) = \langle M_{loop}, \varepsilon\rangle$.

($A_{\mathsf{TM}} \leq_m \text{HALT}_{\mathsf{TM}}$): If x is a string of the form $\langle M, w\rangle$, then $f(x) = \langle M', w\rangle$ where M' simulates M on w, and halts if and only if M accepts. If x is not of the correct form, then $f(x) = \langle M_{loop}, \varepsilon\rangle$. ■

Corollary 5.5.1 $\text{HALT}_{\mathsf{TM}}$ is mapping equivalent (equivalent under mapping reduction) to A_{TM}.

$$\text{HALT}_{\mathsf{TM}} \equiv_m A_{\mathsf{TM}}$$

Hence,

$$\begin{aligned} \text{HALT}_{\mathsf{TM}} &\in \mathsf{TR} \\ &\notin \text{co-}\mathsf{TR} \\ &\notin \mathsf{D} \end{aligned}$$

5.5.2 EQ

Consider the language

$$\text{EQ}_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are Turing machines and } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}$$

We show that

$$A_{\mathsf{TM}} \leq_m \text{EQ} \qquad \text{and} \qquad A_{\mathsf{TM}} \leq_m \overline{\text{EQ}}$$

Proof. Given $\langle M, w\rangle$, if M accepts w, we want to output M_1, M_2 such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.

Let M_1 be the TM which accepts all inputs. $\mathcal{L}(M_1) = \Sigma^*$.

Let M_2 be the TM which on any input, runs M on w and accepts if and only if M accepts.

So, $f(\langle M, w\rangle) = \langle M_1, M_2\rangle$.

$$\mathcal{L}(M_2) = \begin{cases} \Sigma^* & \text{if } M \text{ accepts } w \\ \emptyset & \text{otherwise} \end{cases}$$

Reduction from A_{TM} to $\overline{\mathrm{EQ}}$. ■

Let A be a set of TM descriptions $\{\langle M_1 \rangle, \langle M_2 \rangle, \ldots\}$. Show that if $A \in \mathsf{TR}$, then there is some set of TM descriptions $B \in \mathsf{D}$ such that the set of associated languages of A and B are the same. Formally,

$$\{\mathcal{L}(M) \mid \langle M \rangle \in A\} = \{\mathcal{L}(M) \mid \langle M \rangle \in B\}$$

Let E_B be an enumerator for B.

```
prev-length = 0
for ⟨M_1⟩, ⟨M_2⟩, ... from E_A
    M' = PAD(⟨M_1⟩, prev-length + 1)
    prev-length = |⟨M'⟩|
    print ⟨M'⟩
```

$\mathcal{L}(E_B)$ is the decidable language that is the same as the language recognized by Turing machines in A.

Let $A \in \mathsf{TR}$ be a recognizable set of decider descriptions. Show that there is some decidable set B such that B is not the language of any TM in A.

Let $A = \{\langle D_1 \rangle, \langle D_2 \rangle, \ldots\}$. We want to show that there exists some B such that $\forall i. B \notin \mathcal{L}(D_1)$. Let $\langle D_1 \rangle, \langle D_2 \rangle, \ldots$ be an enumeration of A. Also, let $w_1, w_2, \ldots$ be an enumeration of Σ^* in standard string order.

Let $B = \{w_i \mid w_i \notin \mathcal{L}(D_i)\}$. We first show that B is decidable. We can construct a decider for B as follows.

$D_B(x)$

```
determine i such that w_i = x
run the enumerator for A to get ⟨D_i⟩
simulate D_i on w_i = x
if D_i accepts
    reject
else
    accept
```

We then show that $B \notin \mathcal{L}(D_i)$ for some $\langle D_i \rangle \in A$. If $w_i \in B$, then $w_i \notin \mathcal{L}(D_i)$, and $B \neq \mathcal{L}(D_i)$. If $w_i \notin B$, then $w_i \in \mathcal{L}(D_i)$ and $B \neq \mathcal{L}(D_i)$.

A language A is recognizable if and only if there is a decider V such that for all $x \in \Sigma^*$,

$$x \in A \iff \exists y. V \text{ accepts } (x,y)$$

Think V as a verifier and y as a "certificate" that $x \in A$.

L is recognizable by TM M.

$V(x,y)$

simulate M on x for $|y|$ steps
if M accepts
    accept
elseif M rejects or has not completed
    reject

If $x \in A$, then M accepts within a finite number of steps k. Then, $V(x, y = 1^k)$.

Now suppose that there exists a decider V such that for all $x \in A$, there exists $y \in \Sigma^*$ such that V accepts (x,y).

We need to find a recognizer for A.

$M(x)$

for $y \in \Sigma^*$
    run $V(x,y)$
    if V accepts
        accept
    else
        continue

If $x \in A$, there exists y such that $V(x,y)$ accepts, then y eventually show up in the standard string order of Σ^* and when y is run on V, V accepts. Otherwise, M loops.

5.6 Rice's Theorem

Using reduction, we can prove a really strong theorem about the decidability of certain languages, which can cover many of the languages that we have discussed so far.

Theorem 5.6.1 — Rice's Theorem. Suppose P is a language of Turing machine descriptions such that

1. P is nontrivial: it contains some but not all Turing machine descriptions;
2. P is a ***property of the Turing machine's languages*** (instead of a property of the Turing

machine itself): whenever $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, then $\langle M_1 \rangle \in P \iff \langle M_2 \rangle \in P$.

Then, P is decidable.

Here are some example languages that are covered by Rice's theorem.

Example 5.3

- $\text{EMPTY}_{\text{TM}} = \{\langle M \rangle \mid \mathcal{L}(M) = \emptyset\}$
- $\text{FINITE}_{\text{TM}} = \{\langle M \rangle \mid \mathcal{L}(M) \text{ is finite}\}$
- $\text{DECIDABLE}_{\text{TM}} = \{\langle M \rangle \mid \mathcal{L}(M) \text{ is decidable}\}$

Rice's theorem claims that all of these languages are undecidable.

Note that Rice's theorem does not apply to $\{\langle M \rangle \mid M \text{ is a decider}\}$ because the property being satisfied by the TM descriptions in this language is a property of the Turing machine itself, not its language. Criterion 2 of Rice's theorem is not satisfied. ■

Now, let's prove Rice's theorem using reduction.

Proof. Let T_0 be the TM which always rejects. Without loss of generality, suppose that $\langle T_0 \rangle \notin P$. Otherwise, we can consider $\overline{P}$ which also satisfies the criteria for Rice's theorem.

Since P is nontrivial, there exists TM T_1 such that $\langle T_1 \rangle \in P$. We will show that $\text{A}_{\text{TM}} \leq_m P$. To this end, we map $\langle M, w \rangle$ to M_w where

M_w = "on input x

1. Simulate M on w
2. If M rejects w, reject
3. If M accepts w, then run T_1 on x
   M_w accepts/rejects/loops according to what T_1 does on input x"

$\mathcal{L}(M_w)$ is either $\emptyset = \mathcal{L}(T_0)$ or the langauge of T_1, $\mathcal{L}(T_1)$, depending on whether w is accepted by M. We use M_w's ability to decide between T_0 and T_1 to decide A_{TM}.

Claim. $\langle M, w \rangle \in \text{A}_{\text{TM}} \iff \langle M_w \rangle \in P$.

Let R be the decider for P.

S = "on input $\langle M, w \rangle$

1. Run R on $\langle M, w \rangle$ to check if $\langle M_w \rangle \in P$.
2. If yes, accept; otherwise, reject "

If M accepts w, then $\mathcal{L}(M_w) = \mathcal{L}(T_1)$, so $\langle M_w \rangle \in P$.
If M rejects w, then $\mathcal{L}(M_w) = \emptyset = \mathcal{L}(T_0)$, so $\langle M_w \rangle \notin P$.
If M loops on w, M_w loops on all inputs so $\mathcal{L}(M_w) = \emptyset = \mathcal{L}(T_0)$. ■

Chapter 6 Computation History Method

6.1 Configuration and Computation History

Recall that the ***configuration*** of a Turing machine can be expressed as a triple (q, p, t) where q is the current state, p is the head position, and t is the tape content. Equivalently, this can be written as an encoding t_1qt_2 where $t = t_1t_2$ and the head position is on the first symbol of t_2.

■ **Example 6.1 — Configuration of a Turing Machine.** For example, the configuration

$$(q_3, 6, aaaaaabbbbb)$$

means the Turing machine is in state q_3, with the head pointing at the 6th symbol, and the current tape content being $aaaaaabbbbb$. This can be expressed in a more compact form as

$$aaaaaq_3abbbbb$$

■

A computation history of a Turing machine is a sequence of configurations of the Turing machine until it reaches an accepting state.

Definition 6.1.1 — Computation History. An ***accepting computation history***, or ***computation history***, for a Turing machine M on input w is a sequence of configurations $C_1, C_2, \ldots, C_{accept}$ that M enters until it accepts. Each configuration in this sequence is yielded from the configuration immediately before it.

We encode a computation history as a sequence of configurations separated by the pound sign #.

$$C_1\#C_2\#\ldots\#C_{accept}$$

■ **Example 6.2** A computation history for M on $w = w_1w_2\ldots w_n$ given $\delta(q_0, w_1) = \delta(q_7, a, R)$ and $\delta(q_7, w_2) = (q_8, c, R)$ is as follows

$$\underbrace{q_0w_1w_1\ldots w_n}_{C_1}\#\underbrace{aq_7w_2\ldots w_n}_{C_2}\#\underbrace{acq_8w_3\ldots w_n}_{C_3}\#\cdots\#\underbrace{\ldots q_{accept}\ldots}_{C_{accept}}$$

■

6.2 Decidability of Problems Concerning Linearly Bounded Automata

To understand how to prove undecidability using the computation history method, we first examine a type of machine known as linearly bounded automata (LBA).

Definition 6.2.1 — Linearly Bounded Automaton. A ***linearly bounded automaton (LBA)*** is a single-tape Turing machine that cannot move its head off the input portion of the tape. In other words, the size of the tape is restricted to the size of the input.

6.2.1 A_{LBA} is Decidable

Let

$$A_{\mathsf{LBA}} = \{\langle B, w\rangle \mid \text{LBA } B \text{ accepts } w\}$$

Although it may come as surprising at a first glance, A_{LBA} is, in fact, decidable. This is because of the fac that, for an LBA of input size n, the number of configurations of the LBA is finite, namely equal to $|Q| \times n \times |\Gamma|^n$.

Claim. For inputs of length n, an LBA can only have $|Q| \times n \times |\Gamma|^n$ configurations.

Theorem 6.2.1 A_{LBA} is decidable.

Proof. If B on w runs for too long (more than $|Q| \times n \times |\Gamma|^n$), then by the pigeonhole principle, B must be looping and will never halts or accept. More formally, let us construct a decider for A_{LBA}.

$D_{A_{\mathsf{LBA}}} =$ "on input $\langle B, w\rangle$

1. Let $n = |w|$
2. Run B on w for $|Q| \times n \times |\Gamma|^n$ steps
3. If B has accepted, accept
4. If B has rejected or it is still running, reject"

■

6.2.2 E_{LBA} is Undecidable

Now, let us consider the language

$$E_{\mathsf{LBA}} = \{\langle B\rangle \mid B \text{ is an LBA and } \mathcal{L}(B) = \emptyset\}$$

The next natural question is whether or not E_{LBA} is also decidable. We will show using reduction that A_{TM} can be reduced to E_{LBA} and thus E_{LBA} is undecidable.

Theorem 6.2.2 E_{LBA} is undecidable.

Proof. Assume that $\mathrm{E}_{\mathsf{LBA}}$ is decidable. Then, there exists a Turing machine R that decides $\mathrm{E}_{\mathsf{LBA}}$. We construct a Turing machine S deciding A_{TM}.

$S =$ "on input $\langle M, w\rangle$

1. Construct LBA $B_{\langle M,w\rangle}$ that tests whether its input x is an accepting computation history for M on w, and accepts only if x is an accepting computation history
2. Use R to determine whether $\mathcal{L}(B_{\langle M,w\rangle}) = \emptyset$
3. Accept if no. Reject if yes."

More specifically, we define $B_{\langle M,w\rangle}$ as follows.

$B_{\langle M,w\rangle} =$ "on input x

1. Check if x begins $C_1\#$ where C_1 is the start configuration of M on w
2. Check if each C_{i+1} yields from C_i
3. Check if the final configuration is accepting
4. Accept if all checks pass. Otherwise, reject.

Clearly, S accepts if and only if M on w accepts, and it halts on all inputs. Hence, S decides A_{TM}, but since A_{TM} is undecidable, this is a contradiction. ■

6.3 Post Correspondence Problem

A domino has the form $\left[\frac{t}{b}\right]$ where $t, b \in \Sigma^*$.

Given a collection of dominos $P = \left\{\left[\frac{t_1}{b_1}\right], \ldots, \left[\frac{t_k}{b_k}\right]\right\}$, we say there is a ***match*** if we can make a list using the dominos from P (possibly with repetitions) such that the whole string on the top row is equal to the whole string on the bottom row. The ***Post Correspondence Problem (PCP)*** is to determine whether a collection of dominos P has a match. That is, we want to determine if there is a sequence $i_1, i_2, \ldots, i_m \in \{1, 2, \ldots, k\}$ such that $t_{i_1} \cdot t_{i_2} \cdot \ldots \cdot t_{i_m} = b_{i_1} \cdot b_{i_2} \cdot \ldots \cdot b_{i_m}$.

Figure 6.1 shows a set of dominos and a match.

$$P = \left\{\left[\frac{ab}{aba}\right], \left[\frac{aa}{aba}\right], \left[\frac{ba}{aa}\right], \left[\frac{abab}{b}\right]\right\}$$

$$\text{Match:}\quad \begin{array}{cccccccccccccc} a & b & a & a & b & a & a & a & a & b & a & b \\ a & b & a & a & b & a & a & a & a & b & a & b \end{array}$$

Figure 6.1: A set of dominos P and the corresponding match.

Theorem 6.3.1 PCP is undecidable.

The main idea of the proof: reduction $\mathsf{A_{TM}} \leq_m$ PCP using computation histories. We will in fact show two reductions:

$$\mathsf{A_{TM}} \leq_m \text{MPCP} \leq_m \text{PCP}$$

To make the proof simpler, we first restrict ourselves to the Modified Post Correspondence Problem (MPCP). MPCP adds the additional requirement that the match starts with the first domino $\left[\frac{t_1}{b_1}\right]$.

Proof ($\mathsf{A_{TM}} \leq_m$ MPCP). Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ such that $w \in \Sigma^*$.

We are given $\langle M, w\rangle$ as input and must construct a set of dominos P such that P has a match if and only if M accepts on input w. We assume, without loss of generality, M never attempts to move head left when its head is in the leftmost position.

Given $\langle M, w\rangle$, we construct P. To do that, we introduce the following types of dominos. We put each type of dominos in the order as they are introduced below.

Step 1: $\left[\frac{\#}{\#q_0w_1w_2\dots w_n\#}\right]$ or $\left[\frac{\#}{\#q_0\sqcup}\right]$ if $w = \varepsilon$.

This domino is used as the first domino, which simulates the initial configuration of M.

Step 2: For all $a, a' \in \Gamma$ and $q, q' \in Q$ where $q \neq q_{rej}$, if $\delta(q, a) = (q', a', R)$, we put the domino

$$\left[\frac{qa}{a'q'}\right]$$

This domino simulates the transitions where the head moves to the right.

3: For all $a, a', b \in \Gamma$ and $q, q' \in Q$ where $q \neq q_{rej}$, if $\delta(q, a) = (q', a', L)$, we put the domino

$$\left[\frac{bqa}{q'ba'}\right]$$

This domino simulates the left movement of the head.

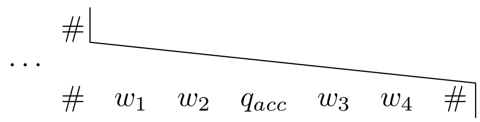
Step 4: For every $a \in \Gamma$, we put $\left[\frac{a}{a}\right]$.

This domino fills in the remaining contents of the tape after a transition.

Step 5: $\left[\frac{\#}{\#}\right]$ and $\left[\frac{\#}{\sqcup\#}\right]$

This domino places the delimiter marking the end of a configuration.

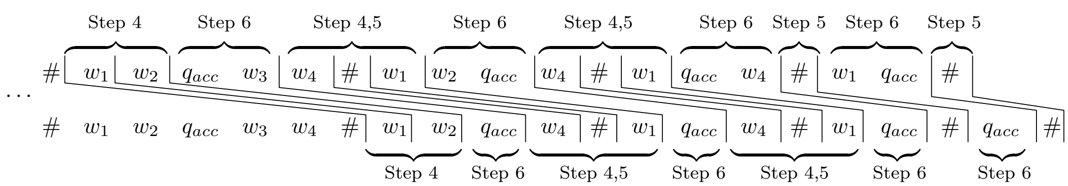
We repeat Step 2-5 to simulate the configurations of M until it reaches the accepting configuration. After all these are finished, we should have a sequence of dominos of the form



Step 6: For every $a \in \Gamma$, put

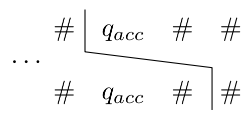
$$\left[\frac{aq_{acc}}{q_{acc}}\right] \quad \text{and} \quad \left[\frac{q_{acc}a}{q_{acc}}\right]$$

This step eats away the symbols adjacent to q_{acc} at the bottom, one at a time, as illustrated below.



Finally,

Step 7: Put the domino $\left[\frac{q_{acc}\#\#}{\#}\right]$. This step finishes off the alignment of the top and bottom row.



This concludes the construction of P.

Claim. P is an accepting instance of MPCP $\iff$ M accepts w.

Next, we provide the construction that shows MPCP $\leq_m$ PCP. It trivially holds that any accepting instance of MPCP is also a PCP. To see why, consider the match $\left[\frac{\#}{\#}\right]$.

Now, we show how to convert an instance of MPCP to PCP such that the converted instance still simulates M on w. Let $P = \left\{\left[\frac{t_1}{b_1}\right], \ldots, \left[\frac{t_k}{b_k}\right]\right\}$ be an input to MPCP. Further, let $\{*, \diamond\}$ be new symbols that are not in any t_i or b_i.

For any string $u = u_1u_2\ldots u_n$, let

$$\circledast u = *u_1 * u_2 \ldots * u_n$$
$$u\circledast = u_1 * u_2 * \ldots u_n *$$
$$\circledast u \circledast = *u_1 * u_2 * \ldots * u_n *$$

Consider the domino set P'

$$P' = \left\{\left[\frac{\circledast t_1}{\circledast b_1 \circledast}\right]\right\} \cup \left\{\left[\frac{\circledast t_i}{b_i \circledast}\right] \mid 1 \leq i \leq k\right\} \cup \left\{\left[\frac{*\diamond}{\diamond}\right]\right\}$$

Then, there is a match in P starting with $\left[\frac{t_1}{b_1}\right]$ if and only if there is a match in P'. The only domino in P' that could start a match is $\left[\frac{\circledast t_1}{\circledast b_1 \circledast}\right]$ because it is the only one where both the top string and bottom string starts with *. ■

Appendix

Commonly Used Axioms & Theorems

Rules of Inference

Axiom 1 — Modus Ponens. $(P \land (P \implies Q)) \implies Q$

Axiom 2 — Modus Tollens. $(\neg Q \land (P \implies Q)) \implies \neg P$

Axiom 3 — Hypothetical Syllogism (transitivity).

$((P \implies Q) \land (Q \implies R)) \implies (P \implies R)$

Axiom 4 — Disjunctive Syllogism. $((P \lor Q) \land \neg P) \implies Q$

Axiom 5 — Addition. $P \implies (P \lor Q)$

Axiom 6 — Simplification. $(P \land Q) \implies P$

Axiom 7 — Conjunction. $((P) \land (Q)) \implies (P \land Q)$

Axiom 8 — Resolution. $((P \lor Q) \land (\neg P \lor R)) \implies (Q \lor R)$

Laws of Logic

Axiom 9 — Implication Law. $(P \implies Q) \equiv (\neg P \lor Q)$

Axiom 10 — Distributive Law.

$$(P \land (Q \lor R)) \equiv ((P \land Q) \lor (P \land R))$$
$$(P \lor (Q \land R)) \equiv ((P \lor Q) \land (P \lor R))$$

Axiom 11 — De Morgan's Law.

$$\neg(P \land Q) \equiv (\neg P \lor \neg Q)$$
$$\neg(P \lor Q) \equiv (\neg P \land \neg Q)$$

Axiom 12 — Absorption Law.

$$(P \lor (P \land Q)) \equiv P$$
$$(P \land (P \lor Q)) \equiv P$$

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

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

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

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

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

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

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

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

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 $d_1, \cdots, d_k$ 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

- $\mathbb{N} = \{0, 1, 2, \ldots\}$: the natural numbers, or non-negative integers. The convention in computer science is to include 0 in the natural numbers.
- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$: the integers
- $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$: the positive integers
- $\mathbb{Z}^- = \{-1, -2, -3, \ldots\}$: 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)
- $\varnothing$ 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 $\mathcal{P}(A) = \big\{\{\}, \{a\}, \{34\}, \{\triangle\}, \{a, 34\}, \{a, \triangle\}, \{34, \triangle\}, \{a, 34, \triangle\}\big\}$.
  $S \in \mathcal{P}(A)$ 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 $\{\ldots, -4, -2, 0, 2, 4, \ldots\} = \{x \in \mathbb{Z} \mid x \text{ is even}\}$.

- $A \times B$: "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 $A \to B$: "the set of all functions from A to B."

- $f : A \to B$ 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: $x \le \max\{x, y\}$
  $y \le \max\{x, y\}$

- $\lfloor x \rfloor$: "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: $x - 1 < \lfloor x \rfloor \le x$
  $\lfloor -x \rfloor = -\lceil x \rceil$
  $\lfloor x + k \rfloor = \lfloor x \rfloor + k$
  $\lfloor \lfloor k/m \rfloor /n \rfloor = \lfloor k/mn \rfloor$
  $(k - m + 1)/m \le \lfloor k/m \rfloor$

- $\lceil x \rceil$: "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: $x \le \lceil x \rceil < x + 1$
  $\lceil -x \rceil = -\lfloor x \rfloor$
  $\lceil x + k \rceil = \lceil x \rceil + k$
  $\lceil \lceil k/m \rceil /n \rceil = \lceil k/mn \rceil$
  $\lceil k/m \rceil \le (k + m - 1)/m$

  Additional property of $\lfloor\ \rfloor$ and $\lceil\ \rceil$: $\lfloor k/2 \rfloor + \lceil k/2 \rceil = k$.

- $|x|$: "absolute value of x" ($|x| = x$ if $x \ge 0$; $-x$ 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: $0 \le m \bmod n < n$
  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 $\{a_n\} = a_1, a_2, \ldots, a_n, \ldots$ *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 $\lim_{n\to\infty} a_n = L$ Otherwise, we say that the sequence *diverges*.

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

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

In particular, if c is any real number, then

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

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

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

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

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

Examples:

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

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

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

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

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

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 IMPLIES B.

    L1. Assume A.
    ⋮
    L2. B
    A IMPLIES B; direct proof: L1, L2

2. **Implication**: Indirect proof of A IMPLIES B.

    L1. Assume $NOT(B)$.
    ⋮
    L2. $NOT(A)$
    A IMPLIES B; indirect proof: L1, L2

3. **Equivalence**: Proof of A IFF B.

    L1. Assume A.
    ⋮
    L2. B
    L3. A IMPLIES B; direct proof: L1, L2
    L4. Assume B.
    ⋮
    L5. A
    L6. B IMPLIES A; direct proof: L4, L5
    A IFF B; equivalence: L3, L6

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

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

5. **Modus Ponens**.

   $\vdots$
   L1. A
   $\vdots$
   L2. A IMPLIES B
   B; modus ponens: L1, L2

6. **Conjunction:** Proof of A AND B:

   $\vdots$
   L1. A
   $\vdots$
   L2. B
   A AND B; proof of conjunction; L1, 2

7. **Use of Conjunction**:

   $\vdots$
   L1. A AND B
   A; use of conjunction: L1
   B; use of conjunction: L1

8. **Implication with Conjunction**: Proof of (A_1 AND A_2) IMPLIES B.

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

9. **Implication with Conjunction**: Proof of A IMPLIES (B_1 AND B_2).

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

10. **Disjunction**: Proof of A OR B and B OR A.

   $\vdots$
   L1. A
   A OR B; proof of disjunction: L1
   B 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 IMPLIES (B_1 OR B_2).

L1. Assume A.
L2. C OR NOT(C) tautology
L3. Case 1: Assume C.
$\vdots$
L4. B_1
L5. B_1 OR B_2; disjunction: L4
L6. C IMPLIES (B_1 OR B_2); direct proof: L3, L5
L7. Case 2: AssumeNOT(C).
$\vdots$
L8. B_2
L9. B_1 OR B_2; disjunction: L8
L10. NOT(C) IMPLIES (B_1 OR B_2); direct proof: L7, L9
L11. B_1 OR B_2; proof by cases: L2, L6, L10
A IMPLIES (B_1 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 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 $\forall x \in D.(P(x)$ IMPLIES $Q(x))$.

    L1. Let x be an arbitrary element of D.
        L2. Assume $P(x)$
        $\vdots$
        L3. $Q(x)$
    L4. $P(x)$ IMPLIES $Q(x)$; direct proof: L2, L3
    Since x is an arbitrary element of D,
    $\forall x \in D.(P(x)$ IMPLIES $Q(x))$; generalization: L1, L4

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

    L1. Assume $\forall x \in D.P(x)$.
    $\vdots$
    L2. $a \in D$
    $P(a)$; specialization: L1, L2
    $\vdots$
    L3. A
    Therefore $(\forall x \in D.P(x))$ IMPLIES A; direct proof: L1, L3

21. **Implication with Universal Quantification**: Proof of A 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 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 $a = \cdots$
    $\vdots$
    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 $a = \cdots$

$\vdots$

L2. $a \in D$

L3. Suppose $P(a)$.

$\vdots$

L4. $Q(a)$

L5. $P(a)$ 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 $(\exists x \in D.P(x))$ IMPLIES A.

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

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

$\vdots$

L2. A

$(\exists x \in D.P(x))$ IMPLIES A; direct proof: L1, L2

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

L1. Assume A.

L2. Let $a = \cdots$

$\vdots$

L3. $a \in D$

$\vdots$

L4. $P(a)$

L5. $\exists x \in D.P(x)$; construction: L2, L3, L4

A 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. $x \in A$ 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. $\forall n \in N.(P(n)$ 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 $\forall j \in N.(j < n$ IMPLIES $P(j))$

$\vdots$

L3. $P(n)$

The following two lines are optional:

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

L5. $\forall n \in N.[\forall j \in N.(j < n$ IMPLIES $P(j))$ 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. $C \neq \phi$; definition: L1, L2
    L4. Let e be the smallest element of C; well ordering principle: L2, L3
        Let $e' = \cdots$
        $\vdots$
        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