École Polytechnique Fédérale de Lausanne

Advanced Information, Computation and Communication I

Notes taken during Prof. Rüdiger Urbanke's lectures at EPFL in Fall 2025.

Elie Baier (397222)
Computer Science Bachelor
November 20, 2025
Version: `35fb1f9`

Contents

Chapter 1

Propositional Logic

1.1 Basic Definitions

Definition 1.1.1 (Proposition).
A proposition is a declarative sentence that is either true or false, but not both.

Example. Here are some examples of propositions and non-propositions:

- "It is raining." This is a proposition because it can be either true or false.
- "$2 + 2 = 4$." This is a proposition because it is always true.
- "The sky is blue." This is a proposition because it can be either true or false depending on the time of day and weather conditions.
- "$x + 2 = 5$." This is not a proposition because it contains a variable (x) and its truth value depends on the value of x.
- "Close the door!" This is not a proposition because it is an imperative sentence and does not have a truth value.

Definition 1.1.2 (Atomic Proposition).
An atomic proposition is a proposition that cannot be broken down into simpler propositions. It is a basic building block of propositional logic.

Proposition can be expressed using variables, typically denoted by letters such as p, q, r, etc. Furthermore, a proposition that is always true is denoted by T, while a proposition that is always false is denoted by F.

Definition 1.1.3 (Compound Proposition).
A compound proposition is a proposition that is formed by combining two or more atomic propositions using logical connectives. The most common logical connectives are:

- Negation ($\neg$): The negation of a proposition p is denoted by $\neg p$.
- Conjunction ($\wedge$): The conjunction of two propositions p and q is denoted by $p \wedge q$.

- Disjunction ($\vee$): The disjunction of two propositions p and q is denoted by $p \vee q$.
- Implication ($\rightarrow$): The implication of two propositions p and q is denoted by $p \rightarrow q$.
- Biconditional ($\leftrightarrow$): The biconditional of two propositions p and q is denoted by $p \leftrightarrow q$.

1.1.1 Truth Tables

Definition 1.1.4 (Truth Table).
A truth table is a mathematical table used to determine the truth value of a compound proposition based on the truth values of its atomic propositions. It lists all possible combinations of truth values for the atomic propositions and shows the resulting truth value of the compound proposition for each combination.

Example. Consider the compound proposition $p \wedge q$, where p and q are atomic propositions. The truth table for this compound proposition is as follows:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

In this truth table, the first two columns represent all possible combinations of truth values for the atomic propositions p and q. The third column shows the resulting truth value of the compound proposition $p \wedge q$ for each combination.

1.2 Logical Connectives

1.2.1 Negation and Conjunction

Definition 1.2.1 (Negation).
The negation of a proposition p is denoted by $\neg p$ (also $\bar{p}$), read "not p". It is true when p is false, and false when p is true.

p	$\neg p$
T	F
F	T

In plain English, $\neg p$ can be expressed as "It is not the case that p" or "It is false that p".

Example. Let p be the proposition "It is raining." Then, $\neg p$ is the proposition "It is not the case that it is raining" or more simply "It is not raining."

Definition 1.2.2 (Conjunction).
The conjunction of two propositions p and q is denoted by $p \wedge q$, read "p and q". It is true

when both p and q are true, and false otherwise.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Example. Let p be the proposition "It is raining." and q be the proposition "It is cold." Then, $p \wedge q$ is the proposition "It is raining and it is cold."

1.2.2 Disjunction

Definition 1.2.3 (Disjunction).
The disjunction of two propositions p and q is denoted by $p \vee q$, read "p or q". It is true when at least one of p or q is true, and false when both are false.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Example. Let p be the proposition "It is raining." and q be the proposition "It is cold." Then, $p \vee q$ is the proposition "It is raining or it is cold."

In natural language, "or" has two distinct meanings:

- Inclusive or: This is the meaning used in propositional logic, where "or" means "at least one of the statements is true". For example, "Candidates for this position should have a degree in mathematics or computer science" means you can have either one or both.
- Exclusive or: This meaning implies that only one of the statements can be true, but not both. For example, "Soup or salad comes with this entrée" means you must choose one and only one.

In propositional logic, we always use the inclusive or.

Definition 1.2.4 (XOR).
The exclusive or of two propositions p and q is denoted by $p \oplus q$, read "p xor q". It is true when exactly one of p or q is true, and false otherwise.

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

In plain English, $p \oplus q$ can be expressed as "Exactly one of p or q is true".

1.2.3 Implication

Definition 1.2.5 (Implication).
The implication of two propositions p and q is denoted by $p \to q$, read "if p then q" or "p implies q". It is false when p is true and q is false, and true otherwise.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

In plain English, $p \to q$ can be expressed as "If p is true, then q must also be true".

An implication does not require any connection between the premise and the conclusion.
A common error is to think that $p \to q$ is the same as $q \to p$. This is not true in general.

Example. A simple way to understand implications is to think of an obligation or contract:

Politician: "If I am elected, I will lower taxes."

If the politician is elected (p is true) and does not lower taxes (q is false), they have broken their promise (the implication is false).
If the politician is not elected (p is false), then no one cares.

In mathematics, $p \to q$ is often read as "p is a sufficient condition for q", or "q is a necessary condition for p".

Example. Let's consider the difference between sufficient and necessary conditions with an example:

- Sufficient condition: "If it is raining, then the ground is wet." Here, raining is a sufficient condition for the ground being wet. If it is raining, we can be sure that the ground is wet. However, the ground can also be wet for other reasons (e.g., someone watering the garden).
- Necessary condition: "If the ground is wet, then it has rained." Here, the ground being wet is a necessary condition for it having rained. If the ground is not wet, we can be sure that it has not rained. However, the ground can be wet for other reasons (e.g., someone watering the garden).

A condition that is both necessary and sufficient is called a **biconditional**.

1.2.4 Biconditional

Definition 1.2.6 (Biconditional).
The biconditional of two propositions p and q is denoted by $p \leftrightarrow q$, read "p if and only if q"

or "p iff q". It is true when both p and q have the same truth value, and false otherwise.

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

In plain English, $p \leftrightarrow q$ can be expressed as "p is true if and only if q".

In English, "if and only if" is often abbreviated as "iff" and it has many alternative phrasings, such as:

- "p is necessary and sufficient for q"
- "p is equivalent to q"
- "if p then q, and conversely"
- "p exactly when q"
- "p just in case q"

In natural language the biconditional is often implicit in statements like "You can drive a car if you have a driving license", which means "You can drive a car if and only if you have a driving license".

1.2.5 Precedence of Logical Connectives

Definition 1.2.7 (Precedence of Logical Connectives).
The precedence of logical connectives is the order in which they are evaluated in a compound proposition. This order determines how the proposition is interpreted and evaluated. The precedence of logical connectives from highest to lowest is as follows:

- Negation ($\neg$)
- Conjunction ($\wedge$)
- Disjunction ($\vee$)
- Implication ($\rightarrow$)
- Biconditional ($\leftrightarrow$)

If needed, parentheses can be used to explicitly indicate the order of evaluation.

Example. Consider the compound proposition $p \vee q \wedge \neg r$. According to the precedence of logical connectives, we first evaluate the negation, then the conjunction, and finally the disjunction. Therefore, the proposition is interpreted as $p \vee (q \wedge (\neg r))$.

1.2.6 Translation between English and Logic

Example. Let p be the proposition "It is below freezing." and q be the proposition "It is snowing.", to express "It is snowing if and only if it is below freezing." in propositional logic, we can write:

$$p \leftrightarrow q$$

Example. Let p be the proposition "you drive over 50 km/h in Lausanne." and q be the proposition "you get a speeding ticket.", to express "You drive over 50 km/h in Lausanne despite not getting a speeding ticket." in propositional logic, we can write:

$$p \wedge \neg q$$

1.3 Classification, Satisfiability and Equivalence of Propositions

1.3.1 Classification of Propositions

Definition 1.3.1 (Tautology).
A tautology is a compound proposition that is always true, regardless of the truth values of its atomic propositions.

Example. The proposition $p \vee \neg p$ is a tautology because it is always true, regardless of the truth value of p.

p	$p \vee \neg p$
T	T
F	T

Definition 1.3.2 (Contradiction).
A contradiction is a compound proposition that is always false, regardless of the truth values of its atomic propositions.

Example. The proposition $p \wedge \neg p$ is a contradiction because it is always false, regardless of the truth value of p.

p	$p \wedge \neg p$
T	F
F	F

Definition 1.3.3 (Contingency).
A contingency is a compound proposition that is neither a tautology nor a contradiction. Its truth value depends on the truth values of its atomic propositions.

Example. The proposition p is a contingency because its truth value depends on the truth value of p.

Example. Let's show that the proposition $(p \wedge q) \to (p \vee q)$ is a tautology.

p	q	$p \wedge q$	$p \vee q$	$(p \wedge q) \to (p \vee q)$
T	T	T	T	T
T	F	F	T	T
F	T	F	T	T
F	F	F	F	T

Since the last column is always true, the proposition is a tautology.

1.3.2 Satisfiability of Propositions

Definition 1.3.4 (Satisfiable Proposition).
A proposition is satisfiable if there exists at least one assignment of truth values to its atomic propositions that makes the compound proposition true. If no such assignment exists, the proposition is unsatisfiable (a contradiction).

Example. Let's determine if the proposition $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$ is satisfiable.

p	q	r	$p \vee \neg q$	$q \vee \neg r$	$r \vee \neg p$	$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$
T	T	T	T	T	T	T
T	T	F	T	F	F	F
T	F	T	T	T	F	F
T	F	F	T	T	T	T
F	T	T	F	T	T	F
F	T	F	F	F	T	F
F	F	T	T	T	T	T
F	F	F	T	T	T	T

Since there are assignments that make the proposition true (for example, $p = T,\ q = T,\ r = T$), the proposition is satisfiable.
Another easier way to determine satisfiability is to manually try to find an assignment that makes the proposition true. For example, we can set $p = T$, $q = T$, and $r = T$ to make the proposition true avoiding the construction of a truth table which can be cumbersome for propositions with many variables (2^n rows for n variables).

1.3.3 Equivalence of Propositions

Definition 1.3.5 (Logical Equivalence).
Two propositions p and q are said to be equivalent, denoted $p \equiv q$, if they have the same truth value in every possible scenario (or if $p \to q$ is a tautology).

Example. Let's show that the propositions $p \to q$ and $\neg p \lor q$ are equivalent.

p	q	$p \to q$	$\neg p$	$\neg p \lor q$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Since the columns for $p \to q$ and $\neg p \lor q$ are identical, the propositions are equivalent.

1.3.4 Logical Equivalences

Non exhaustive list of important logical equivalences:

- Identity Laws:

$$p \land T \equiv p$$
$$p \lor F \equiv p$$

- Domination Laws:

$$p \lor T \equiv T$$
$$p \land F \equiv F$$

- Idempotent Laws:

$$p \lor p \equiv p$$
$$p \land p \equiv p$$

- Double Negation Law:

$$\neg(\neg p) \equiv p$$

- Commutative Laws:

$$p \lor q \equiv q \lor p$$
$$p \land q \equiv q \land p$$

- Associative Laws:

$$(p \lor q) \lor r \equiv p \lor (q \lor r)$$
$$(p \land q) \land r \equiv p \land (q \land r)$$

- Distributive Laws:

$$p \land (q \lor r) \equiv (p \land q) \lor (p \land r)$$
$$p \lor (q \land r) \equiv (p \lor q) \land (p \lor r)$$

- De Morgan's Laws:

$$\neg(p \land q) \equiv \neg p \lor \neg q$$
$$\neg(p \lor q) \equiv \neg p \land \neg q$$

- Absorption Laws:

$$p \lor (p \land q) \equiv p$$
$$p \land (p \lor q) \equiv p$$

- Negation Laws:

$$p \vee \neg p \equiv T$$
$$p \wedge \neg p \equiv F$$

More equivalences laws can be found on moodle (or in the cheetsheet for the exam).

Example. Since we know that $p \to q \equiv \neg q \to \neg p$ we also know that, for example, $(p_1 \vee p_2) \to (q_1 \wedge q_2) \equiv \neg(q_1 \wedge q_2) \to \neg(p_1 \vee p_2)$.

1.3.5 Contrapositive, Converse, and Inverse

Definition 1.3.6 (Contrapositive).
The contrapositive of an implication $p \to q$ is the implication $(\neg q) \to (\neg p)$. An implication is logically equivalent to its contrapositive.

Definition 1.3.7 (Converse).
The converse of an implication $p \to q$ is the implication $q \to p$. An implication is not logically equivalent to its converse in general.

Definition 1.3.8 (Inverse).
The inverse of an implication $p \to q$ is the implication $\neg p \to \neg q$. An implication is not logically equivalent to its inverse in general.

1.3.6 Equivalence Proofs

To prove that $A \equiv B$, we use a sequence of logical equivalences starting from A and ending at B (or vice versa). Each step in the sequence must be justified by a known logical equivalence.

Example. Let's show that $\neg(p \vee (\neg p \wedge q)) \equiv \neg p \wedge \neg q$.

$$\begin{aligned}
\neg(p \vee (\neg p \wedge q)) &\equiv \neg p \wedge \neg(\neg p \wedge q) && \text{(De Morgan's Law)} \\
&\equiv \neg p \wedge (\neg\neg p \vee \neg q) && \text{(De Morgan's Law)} \\
&\equiv \neg p \wedge (p \vee \neg q) && \text{(Double Negation)} \\
&\equiv (\neg p \wedge p) \vee (\neg p \wedge \neg q) && \text{(Distributive Law)} \\
&\equiv F \vee (\neg p \wedge \neg q) && \text{(Negation Law)} \\
&\equiv \neg p \wedge \neg q && \text{(Identity Law)}
\end{aligned}$$

Therefore, we have shown that $\neg(p \vee (\neg p \wedge q)) \equiv \neg p \wedge \neg q$.

1.3.7 Methods to Prove Tautology, Contingency or Contradiction

To prove that an expression is a tautology, contingency or contradiction we can use one of the following methods:

- Truth Table (most straightforward but can be cumbersome for many variables).

- Equivalence Proof (shorter but requires knowledge of equivalence laws and intuition).
- Counter Example (shorter but requires intuition).

1.3.8 Do We Need All Propositions?

Definition 1.3.9 (Functionally Complete Set of Connectives)**.**
A set of logical connectives is functionally complete if every compound proposition can be expressed using only the connectives in that set.

Example. The set of connectives $\{\neg, \wedge, \vee\}$ is functionally complete because we can express all other connectives using only negation and conjunction. For example:

- Implication: $p \to q \equiv \neg p \vee q \equiv \neg(\neg p \wedge \neg q)$
- Biconditional: $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \equiv (\neg p \vee q) \wedge (\neg q \vee p)$

Therefore, we can express any compound proposition using only negation and conjunction.

1.4 Normal Forms

Definition 1.4.1 (Normal Form)**.**
A normal form is a standardized way of expressing a compound proposition that can be used in automated proofing of theorems.

The two most common normal forms are the disjunctive normal form (DNF) and the conjunctive normal form (CNF).

1.4.1 Disjunctive Normal Form

Definition 1.4.2 (Disjunctive Normal Form)**.**
A compound proposition is in disjunctive normal form (DNF) if it is a disjunction of one or more conjunctions of one or more midterms (or literals). A midterm is either an atomic proposition or its negation.

Example. The following propositions are in (full) disjunctive normal form:

- $(p \wedge q) \vee (\neg p \wedge r)$ (full DNF)
- $(\neg p \wedge q) \vee (p \wedge \neg r)$ (full DNF)
- $p \vee (\neg q \wedge r)$ (not full DNF, because the first term is not a conjunction)

To find the DNF of a proposition, we can use a truth table to identify the rows where the proposition is true, and then construct the DNF by taking the disjunction of the conjunctions of the literals corresponding to those rows.

Example. Find the DNF of the proposition $(p \vee q) \to \neg r$.

p	q	r	$p \vee q$	$\neg r$	$(p \vee q) \to \neg r$
T	T	T	T	F	F
T	T	F	T	T	T
T	F	T	T	F	F
T	F	F	T	T	T
F	T	T	T	F	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	F	T	T

The proposition is true for the rows 2, 4, 6, 7, and 8. Therefore, the DNF is:

$$(p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$$

1.4.2 Conjunctive Normal Form

Definition 1.4.3 (Conjunctive Normal Form).
A compound proposition is in conjunctive normal form (CNF) if it is a conjunction of one or more disjunctions of one or more clauses (or literals). A clause is either an atomic proposition or its negation.

Example. The following propositions are in (full) conjunctive normal form:

- $(p \vee q) \wedge (\neg p \vee r)$ (full CNF)
- $(\neg p \vee q) \wedge (p \vee \neg r)$ (full CNF)
- $p \wedge (\neg q \vee r)$ (not full CNF, because the first term is not a disjunction)

To find the CNF of a proposition, we can use a truth table to identify the rows where the proposition is false, and then construct the CNF by taking the conjunction of the disjunctions of the literals corresponding to those rows.

Example. Find the CNF of the proposition $(p \vee q) \to \neg r$.

p	q	r	$p \vee q$	$\neg r$	$(p \vee q) \to \neg r$
T	T	T	T	F	F
T	T	F	T	T	T
T	F	T	T	F	F
T	F	F	T	T	T
F	T	T	T	F	F
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	F	T	T

The proposition is false for the rows 1, 3, and 5. Therefore, the CNF is:

$$(\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (p \vee \neg q \vee \neg r)$$

1.4.3 Relation between DNF and CNF

Let now take the same example that will show the link between DNF and CNF.

Example. If we take the DNF of the same proposition that was used above but we negate it, it becomes $\neg((p \vee q) \to \neg r)$ and the truth table is:

p	q	r	$p \vee q$	$\neg r$	$(p \vee q) \to \neg r$	$\neg((p \vee q) \to \neg r)$
T	T	T	T	F	F	T
T	T	F	T	T	T	F
T	F	T	T	F	F	T
T	F	F	T	T	T	F
F	T	T	T	F	F	T
F	T	F	T	T	T	F
F	F	T	F	F	T	F
F	F	F	F	T	T	F

The proposition is true for the rows 1, 3, and 5. Therefore, the DNF of $\neg((p \vee q) \to \neg r)$ is:

$$(p \wedge q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)$$

If we now negate this DNF, we get back to the CNF of the original proposition:

$$\neg((p \wedge q \wedge r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)) \equiv (\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (p \vee \neg q \vee \neg r)$$

This demonstrates the connection between DNF and CNF, as the negation of a proposition's DNF corresponds to the CNF of its negation.

1.4.4 Finding DNF and CNF without Truth Table

To find the DNF and CNF of a proposition without using a truth table, we can use logical equivalences and simplifications. Here is a general recipe:

- Eliminate equivalences and implications
- Move negation inward using De Morgan's Law
- Use distributive and associative laws (for DNF: $a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$, for CNF: $a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$)

Example. Let's put the proposition, $s \to \neg(p \to q)$, in DNF and CNF form without using a truth table:

$$\begin{aligned} s \to \neg(p \to q) &\equiv \neg s \vee \neg(p \to q) && \text{(Implication)} \\ &\equiv \neg s \vee \neg(\neg p \vee q) && \text{(Implication)} \\ &\equiv \neg s \vee (p \wedge \neg q) && \text{(De Morgan's Law)} \end{aligned}$$

The proposition is now in DNF form. If we want to get the full DNF we can substitute the following:

$$(p \wedge \neg q) \equiv (p \wedge \neg q \wedge s) \vee (p \wedge \neg q \wedge \neg s)$$
$$\neg s \equiv (\neg s \wedge p \wedge q) \vee (\neg s \wedge \neg p \wedge q) \vee (\neg s \wedge \neg p \wedge \neg q) \vee (\neg s \wedge p \wedge \neg q)$$

At most we will have 2^n terms in the full DNF where n is the number of atomic propositions but in some cases some terms will be the same.

Now if we want to get the CNF form, we can continue:

$$\neg s \vee (p \wedge \neg q) \equiv (\neg s \vee p) \wedge (\neg s \vee \neg q) \qquad \text{(Distributive Law)}$$

The proposition is now in CNF form.

1.5 Exercices

This section gathers a selection of exercises related to Chapter 1, taken from weekly assignments, past exams, textbooks, and other sources. The origin of each exercise will be indicated at its beginning.

Chapter 2

Predicate Logic

In propositional logic, some expression cannot be expressed easily or cannot be expressed at all. For example, the statement "x + 2 = 5." is not a proposition because it contains a variable x. However, we can express this statement as a predicate.

2.1 Propositional Functions

2.2 Quantifiers

2.2.1 Universal Quantifier

2.2.2 Existential Quantifier

2.2.3 Uniqueness Quantifier

2.2.4 Precedence, Scope and Binding

2.2.5 Validity and Satisfiability

2.3 Exercices

This section gathers a selection of exercises related to Chapter 2, taken from weekly assignments, past exams, textbooks, and other sources. The origin of each exercise will be indicated at its beginning.

Exercise 2.3.1 (*Explique AI*)**.**
Given these two assemptions:

- "Running is not difficult or not many students like running"
- "If jumping is easy, then running is not difficult"

How many of the following are valid conclusion of these assemptions?

- "Jumping is not easy, if many students like running"

- ”Jumping is not easy or running is difficult”
- ”If not many students like running, then either jumping is not easy or running is not difficult”
- ”Running is not difficult or jumping is not easy”

Answer:
Let’s translate the propositions as following:

- D: ”Running is difficult”
- M: ”Many student like running”
- J: ”Jumping is easy”

And the premises translate to:

- $D \vee \neg M$
- $J \to \neg D \equiv \neg J \vee \neg D$

1. If M holds, premise (1) forces D (because $D \vee \neg M$ and $\neg M$ is false). From the contrapositive of (2) we have $D \to \neg J$. So $M \to D$ and $D \to \neg J$ gives $M \to \neg J$. **Valid.**

Chapter 3

Methods of Proof

Definition 3.0.1 (Argument).
An argument consists of a set of premises and a conclusion. The premises are propositions that are assumed to be true, and the conclusion is a proposition that is inferred from the premises. An argument is often written in the following form:

$$\begin{array}{l} P_1 \\ P_2 \\ \ldots \\ P_n \\ \hline \therefore \quad Q \end{array}$$

where $P_1, P_2, ..., P_n$ are the premises and Q is the conclusion.

Definition 3.0.2 (Argument Form).
An argument form is a template for an argument that uses propositional variables instead of specific propositions. For example, the argument form for modus ponens is:

$$\begin{array}{l} p \to q \\ p \\ \hline \therefore \quad q \end{array}$$

where p and q are propositional variables that can be replaced with any propositions.

Definition 3.0.3 (Valid Argument).
An argument is valid if and only if the conclusion is true whenever all the premises are true.

3.1 Inference Rules

Definition 3.1.1 (Inference Rule).
An inference rule is a valid argument form that can be used to derive a conclusion from a set of premises.

3.1.1 Rules of Inference for Propositions

Definition 3.1.2 (Modus Ponens).
If $P \to Q$ is true and P is true, then Q must be true. It is often written as follows:

$$\begin{array}{ll} & P \to Q \\ & P \\ \hline \therefore & Q \end{array}$$

It can also be written as a tautology: $(P \wedge (P \to Q)) \to Q$.

Definition 3.1.3 (Conjunction).
If P is true and Q is true, then $P \wedge Q$ is true. It is often written as follows:

$$\begin{array}{ll} & P \\ & Q \\ \hline \therefore & P \wedge Q \end{array}$$

It can also be written as a tautology: $((P) \wedge (Q)) \to (Q \wedge P)$.

Definition 3.1.4 (Modus Tollens).
If $P \to Q$ is true and $\neg Q$ is true, then $\neg P$ must be true. It is often written as follows:

$$\begin{array}{ll} & P \to Q \\ & \neg Q \\ \hline \therefore & \neg P \end{array}$$

It can also be written as a tautology: $(\neg Q \wedge (P \to Q)) \to \neg P$.

Definition 3.1.5 (Hypothetical Syllogism).
If $P \to Q$ is true and $Q \to R$ is true, then $P \to R$ must be true. It is often written as follows:

$$\begin{array}{ll} & P \to Q \\ & Q \to R \\ \hline \therefore & P \to R \end{array}$$

It can also be written as a tautology: $((P \to Q) \wedge (Q \to R)) \to (P \to R)$.

Example. Let $p =$ "I have passed AICC", $q =$ "I can advance to year 2 of studies" and the premises be:

- "I have passed AICC. I can advance to year 2 of studies." $(p \to q)$
- "I cannot advanced to year 2 of studies." $(\neg q)$

We can represent the argument as follows:

$$\begin{array}{l} p \to q \\ \neg q \\ \hline \therefore \quad \neg p \end{array}$$

We can conclude that "I have not passed AICC." ($\neg p$) by using modus tollens.

Definition 3.1.6 (Disjunctive Syllogism).
If $P \vee Q$ is true and $\neg P$ is true, then Q must be true. It is often written as follows:

$$\begin{array}{l} P \vee Q \\ \neg P \\ \hline \therefore \quad Q \end{array}$$

It can also be written as a tautology: $((P \vee Q) \wedge \neg P) \to Q$.

Definition 3.1.7 (Addition).
If P is true, then $P \vee Q$ is true. It is often written as follows:

$$\begin{array}{l} P \\ \hline \therefore \quad P \vee Q \end{array}$$

It can also be written as a tautology: $P \to (P \vee Q)$.

Definition 3.1.8 (Simplification).
If $P \wedge Q$ is true, then P is true. It is often written as follows:

$$\begin{array}{l} P \wedge Q \\ \hline \therefore \quad P \end{array}$$

It can also be written as a tautology: $(P \wedge Q) \to P$.

Definition 3.1.9 (Resolution).
If $P \vee Q$ is true and $\neg P \vee R$ is true, then $Q \vee R$ must be true. It is often written as follows:

$$\begin{array}{l} P \vee Q \\ \neg P \vee R \\ \hline \therefore \quad Q \vee R \end{array}$$

It can also be written as a tautology: $((P \vee Q) \wedge (\neg P \vee R)) \to (Q \vee R)$.

Example. In the sentence "It is below freezing now. Therefore, it is below freezing or raining now", let $p =$ "It is below freezing now" and $q =$ "It is raining now". We can represent the

argument as follows:

$$\begin{array}{l} p \\ \hline \therefore \quad p \vee q \end{array}$$

Since this argument follows the form of addition, we can conclude that "It is below freezing or raining now." $(p \vee q)$.

Example. In the sentence "If it rains today, we will not have a barbecue today. If we do not have a barbecue today, then we will have a barbecue tomorrow", let $p =$ "It rains today", $q =$ "We will have a barbecue today" and $r =$ "We will have a barbecue tomorrow". We can represent the argument as follows:

$$\begin{array}{l} p \to \neg q \\ \neg q \to r \\ \hline \therefore \quad p \to r \end{array}$$

Since this argument follows the form of hypothetical syllogism, we can conclude that "If it rains today, then we will have a barbecue tomorrow." $(p \to r)$.

Note that even seemingly "obvious" conclusions imply an argument.

Example. From $p \wedge (p \to q)$, we can conclude q:

$$\begin{array}{l} p \wedge (p \to q) \\ \hline \therefore \quad q \end{array}$$

This can be simplified to:

$$\begin{array}{l} p \\ p \to q \\ \hline \therefore \quad q \end{array}$$

3.1.2 Rules of Inference for Quantified Statements

Definition 3.1.10 (Universal Instantiation).
If $P(x)$ is a predicate and c is an element in the domain of discourse, then from $\forall x P(x)$ we can conclude $P(c)$. It is often written as follows:

$$\begin{array}{l} \forall x P(x) \\ \hline \therefore \quad P(c) \end{array}$$

Definition 3.1.11 (Universal Generalization).
If $P(c)$ is true for an arbitrary element c in the domain of discourse, then we can conclude $\forall x P(x)$. It is often written as follows:

$$\begin{array}{l} P(c) \\ \hline \therefore \quad \forall x P(x) \end{array}$$

Note that c must be arbitrary, meaning that it cannot have any special properties that distinguish it from other elements in the domain.

Definition 3.1.12 (Existential Instantiation)**.**
If $P(x)$ is a predicate and c is an element in the domain of discourse, then from $\exists x P(x)$ we can conclude $P(c)$. It is often written as follows:

$$\begin{array}{ll} & \exists x P(x) \\ \hline \therefore & P(c) \end{array}$$

Note that c must be a new element that does not appear elsewhere in the argument.

Definition 3.1.13 (Existential Generalization)**.**
If $P(c)$ is true for some element c in the domain of discourse, then we can conclude $\exists x P(x)$. It is often written as follows:

$$\begin{array}{ll} & P(c) \\ \hline \therefore & \exists x P(x) \end{array}$$

Example. Let the domain of discourse be all students in a class, let A be a student in the class" and let the predicate $P(x) =$ "x has taken a course in Java". Given the premise "All students in the class have taken a course in Java." ($\forall x P(x)$), we can represent the argument as follows:

$$\begin{array}{ll} & \forall x P(x) \\ \hline \therefore & P(Sara) \end{array}$$

where Sara is a student in the class. By using universal instantiation, we can conclude that "Sara has taken a course in Java." ($P(Sara)$).

Example. Let's use the rules of inference to construct a valid argument showing that "Someone who passed the first exam has not read the book". Let's define some predicates as follows:

- $P(x)$: "x passed the first exam"
- $B(x)$: "x has read the book"
- $C(x)$: "x is in this class"

The the conslusion can be expressed as:

$$\exists x(P(x) \wedge \neg B(x))$$

The premises are:

- "Everyone in this class passed the first exam". ($\forall x(C(x) \rightarrow P(x))$)
- "A student in this class has not read the book". ($\exists x(C(x) \wedge \neg B(x))$)

We can represent the argument as follows:

$$\begin{array}{ll} & \forall x(C(x) \to P(x)) \\ & \exists x(C(x) \wedge \neg B(x)) \\ \hline \therefore & \exists x(P(x) \wedge \neg B(x)) \end{array}$$

From the second premise, we can use existential instantiation to introduce a new constant c such that:

$$C(c) \wedge \neg B(c)$$

From this, we can use simplification to obtain:

$$C(c)$$

From the first premise, we can use universal instantiation to obtain:

$$C(c) \to P(c)$$

From this and $C(c)$, we can use modus ponens to obtain:

$$P(c)$$

Finally, we can use conjunction to obtain:

$$P(c) \wedge \neg B(c)$$

And then we can use existential generalization to obtain the conclusion:

$$\exists x(P(x) \wedge \neg B(x))$$

3.2 Terminology for Proofs

Definition 3.2.1 (Mathematical Proof)**.**
A mathematical proof is a finite sequence of statements that starts with the premises and ends with the conclusion, where each statement is either a premise or follows from previous statements by a valid inference rule.

Definition 3.2.2 (Theorem)**.**
A theorem is a mathematical statement that can be shown to be true using:

- Axioms: statements that are assumed to be true without proof
- Definitions: statements that define new concepts or terms
- Previously proven theorems
- Valid inference rules

Example. Let's take some examples of axioms, definitions and theorems:

- Axiom: "For any integer n, $n + 0 = n$."

- Definition: "An integer n is even if there exists an integer k such that $n = 2k$."
- Theorem: "The sum of two even integers is even."

In mathematics, computer science and related fields, informal proofs are often written in a natural language, such as English, with mathematical notation and symbols used to express mathematical concepts and relationships. In these proofs often:

- Use more than one valid inference rule in a single step.
- Might omit steps if they are considered "obvious" or "trivial".
- Use inference rules that are not explicitly stated.
- Are easier to read and understand for humans.

However, the underlying structure of the proof is still based on the principles of logic and valid inference rules.

Definition 3.2.3 (Lemma, Corollary and Conjecture).
A lemma is a proven statement that is used as a stepping stone to prove a larger theorem. A corollary is a statement that follows directly from a theorem or lemma. A conjecture is a statement that is believed to be true but has not yet been proven.

3.3 Proving Theorems

To prove theorems of the form:

$$\forall x(P(x) \to Q(x))$$

we can use the following strategy:

- Assume $P(c)$ for an arbitrary constant c.
- Use valid inference rules to derive $Q(c)$.
- Conclude, by universal generalization, that $\forall x(P(x) \to Q(x))$ is true.

Definition 3.3.1 (Trivial Proof).
A trivial proof is a proof that is considered obvious or self-evident, often because it relies on basic axioms or definitions.

Definition 3.3.2 (Vacuous Proof).
A vacuous proof is a proof that shows a statement is true because the premise is false.

Example. Let $P(n)$ be "If a and b are positive integers with $a \geq b$, then $a^n \geq b^n$", where the domain of discourse is the set of positive integers. We want to prove that $P(0)$ is true. Since $a^0 = 1$ and $b^0 = 1$ for any positive integers a and b, we have $1 \geq 1$, which is true. Therefore, $P(0)$ is true by a trivial proof.

Example. Let's prove that if n is an integer with $10 \leq n \leq 15$ which is a perfect square, then n is also a perfect cube.
There are no integers n such that $10 \leq n \leq 15$ which are perfect squares as $3^2 = 9$ and $4^2 = 16$. Therefore, the hypothesis p is false, and the statement is true by a vacuous proof.

3.3.1 Direct Proofs

Definition 3.3.3 (Direct Proof).
A direct proof is a proof that shows a statement is true by using definitions, axioms, theorems and valid inference rules to derive the conclusion directly from the premises.

Example. Let's prove that if n is an odd integer, then n^2 is also an odd integer.
Let n be an arbitrary odd integer. By definition of odd integers, there exists an integer k such that $n = 2k + 1$. We can then compute:

$$n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$$

Since $2k^2 + 2k$ is an integer, we can conclude that n^2 is also an odd integer by definition of odd integers. Therefore, we have shown that if n is an odd integer, then n^2 is also an odd integer by a direct proof.

3.3.2 Indirect Proofs

Definition 3.3.4 (Proof by Contraposition).
A proof by contraposition is a proof that shows a statement of the form $P \to Q$ is true by proving its contrapositive $\neg Q \to \neg P$ is true.

Example. Let's prove that if n^2 is an odd integer, then n is also an odd integer by contraposition.
The contrapositive of the statement is "If n is not an odd integer, then n^2 is not an odd integer", which is equivalent to "If n is an even integer, then n^2 is an even integer".
Let n be an arbitrary even integer. By definition of even integers, there exists an integer k such that $n = 2k$. We can then compute:

$$n^2 = (2k)^2 = 4k^2 = 2(2k^2)$$

Since $2k^2$ is an integer, we can conclude that n^2 is also an even integer by definition of even integers. Therefore, we have shown that if n^2 is an odd integer, then n is also an odd integer by a proof by contraposition.

Definition 3.3.5 (Proof by Contradiction).
A proof by contradiction is a proof that shows a statement P is true by assuming the negation of the conclusion $\neg Q$ is also true then performing a direct proof to derive a con-

tradiction, for example, in the form of:

$$(P \wedge \neg Q) \rightarrow (R \wedge \neg R) \equiv (P \wedge \neg Q) \rightarrow \text{False}$$

This type of proof works because:

$$(P \wedge \neg Q) \rightarrow \text{False} \equiv \neg(P \wedge \neg Q) \vee \text{False} \equiv \neg(P \wedge \neg Q) \equiv \neg P \vee Q \equiv P \rightarrow Q$$

Example. Let's prove that if n^2 is an odd integer, then n is also an odd integer by contradiction.
We will assume the negation of the conclusion, which is "n is not an odd integer", and show that this leads to a contradiction.
Let n be an arbitrary integer such that n^2 is an odd integer and n is not an odd integer. By definition of odd integers, if n is not an odd integer, then n must be an even integer. Therefore, there exists an integer k such that $n = 2k$. We can then compute:

$$n^2 = (2k)^2 = 4k^2 = 2(2k^2)$$

Since $2k^2$ is an integer, we can conclude that n^2 is an even integer by definition. However, this contradicts our assumption that n^2 is an odd integer. Therefore, we have shown that if n^2 is an odd integer, then n is also an odd integer by a proof by contradiction.

Example. Let's prove that $\sqrt{2}$ is irrational by contradiction.
We will assume the negation of the conclusion, which is "$\sqrt{2}$ is rational", and show that this leads to a contradiction.
If $\sqrt{2}$ is rational, then there exist integers a and b such that $\sqrt{2} = \frac{a}{b}$, where $b \neq 0$ and a and b have no common factors (i.e., the fraction is in lowest terms). We can then compute:

$$2 = \left(\frac{a}{b}\right)^2 = \frac{a^2}{b^2}$$

Multiplying both sides by b^2, we get:

$$2b^2 = a^2$$

This implies that a^2 is even, since it is equal to $2b^2$. By the definition of even integers, this means that a must also be even. Therefore, there exists an integer k such that $a = 2k$. We can then compute:

$$2b^2 = (2k)^2 = 4k^2$$

Dividing both sides by 2, we get:

$$b^2 = 2k^2$$

This implies that b^2 is even, since it is equal to $2k^2$. By the definition of even integers, this means that b must also be even. However, this contradicts our assumption that a and b have no common factors, since both a and b are even. Therefore, we have shown that $\sqrt{2}$ is irrational by a proof by contradiction.

Definition 3.3.6 (Proof by Cases).
A proof by cases is a proof that shows a statement P is true by dividing the problem into a finite number of cases $((p_1 \vee p_2 \vee \ldots \vee p_n) \to Q)$ and proving that each case $p_i \to Q$ is true.

Example. Let's prove that if n is an integer, then $n^2 \geq n$.
We can divide the problem into 3 cases:

- Negative integers $(n < 0)$
- Zero $(n = 0)$
- Positive integers $(n > 0)$

Then we can prove that $n^2 \geq n$ is true for each case:

- If $n < 0$, then n^2 is positive and n is negative, so $n^2 \geq n$ is true.
- If $n = 0$, then $n^2 = 0$ and $n = 0$, so $n^2 \geq n$ is true.
- If $n > 0$, then $n \geq 1 \iff n^2 \geq n$, so $n^2 \geq n$ is true.

Since $n^2 \geq n$ is true for all cases, we can conclude that if n is an integer, then $n^2 \geq n$ by a proof by cases.

Definition 3.3.7 (Without Loss of Generality).
In a proof by cases, we can sometimes assume "without loss of generality" (WLOG) that a certain case holds, if the other cases are symmetric or similar. This means that we can prove the statement for one case and then conclude that it holds for all cases.

Definition 3.3.8 (Counterexample).
A counterexample is an example that shows a statement is false. To disprove a statement of the form $\forall x P(x)$, we can provide a counterexample c such that $\neg P(c)$ is true.

Example. Let's disprove the statement "All prime numbers are odd."
A counterexample is the prime number 2, which is even. Therefore, the statement is false.

3.3.3 Proofs of Equivalences

To prove a statement of the form $P \leftrightarrow Q$, we can use the following strategy:

- Prove $P \to Q$ using a direct proof or an indirect proof.
- Prove $Q \to P$ using a direct proof or an indirect proof.

To prove an existence proof of the form $\exists x P(x)$, we can use one of the following strategies:

- Provide a specific example (or witness) c such that $P(c)$ is true (constructive proof).
- Show that the negation of the statement leads to a contradiction (non-constructive proof).

To prove a uniqueness proof of the form "There exists a unique x such that $P(x)$", we can use the following strategy:

- Prove the existence of such an x using a constructive or non-constructive proof.
- Prove the uniqueness of such an x by assuming there are two such elements x and y and showing that $x = y$.

3.4 Exercices

This section gathers a selection of exercises related to Chapter 3, taken from weekly assignments, past exams, textbooks, and other sources. The origin of each exercise will be indicated at its beginning.

Chapter 4

Sets and Functions

4.1 Sets

Definition 4.1.1 (Set).
A set is a well-defined collection of distinct objects, considered as an object in its own right. Sets are typically denoted by capital letters (e.g., A, B, C), and the objects within a set are called elements or members. The notation $x \in A$ indicates that x is an element of the set A, while $x \notin A$ indicates that x is not an element of A.

The order in a set and duplicate elements do not matter. For example, the sets $\{1, 2, 3\}$, $\{3, 2, 1\}$, and $\{1, 2, 2, 3\}$ are all considered equal.
To describe a set, we can use:

- All members are listed between curly braces, e.g., $A = \{1, 2, 3, 4, 5\} = \{1, \ldots, 5\}$ (Roster form).
- A property or condition that characterizes its members, e.g., $B = \{x \mid P(x)\}$ (Set-builder form).

Example. Some examples of sets:

- The set of all letters in the English alphabet: $A = \{a, b, c, \ldots, z\}$.
- The set of all even natural numbers: $E = \{x \mid x = 2n, n \in \mathbb{N}\}$.
- The set of all prime numbers less than 10: $P = \{2, 3, 5, 7\}$.
- The set of all real numbers between 0 and 1: $R = \{x \mid 0 < x < 1, x \in \mathbb{R}\}$.

Example. Some sets are given special names:

- The empty set (or null set) is the set that contains no elements, denoted by $\emptyset$ or $\{\}$.
- The set of natural numbers is denoted by $\mathbb{N} = \{0, 1, 2, \ldots\}$.
- The set of integers is denoted by $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
- The set of positive integers is denoted by $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$.

- The set of real numbers is denoted by $\mathbb{R}$.
- The set of complex numbers is denoted by $\mathbb{C} = \{a + bi \mid a, b \in \mathbb{R}, i^2 = -1\}$.

Sets can also contain other sets as elements. For example, $A = \{1, 2, \{3, 4\}\}$ is a set containing the elements 1, 2, and the set $\{3, 4\}$.

Definition 4.1.2 (Empty Set).
The empty set is the unique set that contains no elements. It is denoted by $\emptyset$ or $\{\}$. The empty set is a subset of every set, including itself.

Definition 4.1.3 (Singleton Set).
A singleton set is a set that contains exactly one element. For example, $\{a\}$ is a singleton set containing the element a.

Note that the empty set $\emptyset$ is different from the singleton set $\{\emptyset\}$, which contains the empty set as its only element.

Definition 4.1.4 (Universal Set).
The universal set is the set that contains all the objects or elements under consideration, usually denoted by U. The specific elements of the universal set depend on the context of the discussion.

4.1.1 Interval Notation

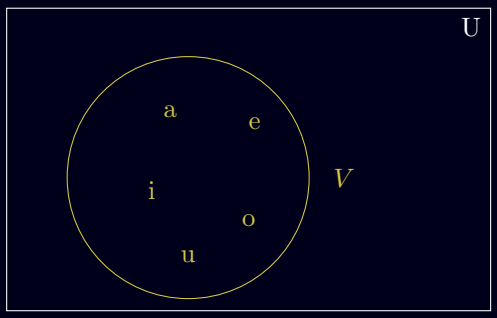
Definition 4.1.5 (Interval).
An interval is a set of real numbers that contains all real numbers between any two numbers in the set. Intervals can be classified into several types based on whether they include or exclude their endpoints:

- Closed interval: $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ (includes both endpoints).
- Open interval: $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$ (excludes both endpoints).
- Half-open (or half-closed) interval: $[a, b) = \{x \in \mathbb{R} \mid a \leq x < b\}$ (includes a but excludes b), and $(a, b] = \{x \in \mathbb{R} \mid a < x \leq b\}$ (excludes a but includes b).
- Infinite intervals: $[a, \infty) = \{x \in \mathbb{R} \mid x \geq a\}$, $(a, \infty) = \{x \in \mathbb{R} \mid x > a\}$, $(-\infty, b] = \{x \in \mathbb{R} \mid x \leq b\}$, and $(-\infty, b) = \{x \in \mathbb{R} \mid x < b\}$.

4.1.2 Venn Diagrams

Definition 4.1.6 (Venn Diagram).
A Venn diagram is a graphical representation of sets and their relationships using overlapping circles or other shapes. Each circle represents a set, and the overlapping regions represent the intersections of the sets. Venn diagrams are useful for visualizing concepts such as union, intersection, and complement of sets.

Example. Here is a simple Venn diagram representing the set V of vowel letters in the English alphabet:



4.1.3 Subsets

Definition 4.1.7 (Subset).
A set A is a subset of a set B (or B is a superset of A), denoted by $A \subseteq B$, if every element of A is also an element of B.

Definition 4.1.8 (Proper Subset).
A set A is a proper subset of a set B, denoted by $A \subset B$, if $A \subseteq B$ and there exists at least one element in B that is not in A. We can write this as:

$$\forall x(x \in A \to x \in B) \land \exists y(y \in B \land y \notin A)$$

Example. Some examples of subsets:

- $\{1, 2\} \subseteq \{1, 2, 3, 4, 5\}$.
- $\{a, e, i\} \subseteq \{a, b, c, d, e, f, g, h, i\}$.
- $\emptyset \subseteq A$ for any set A.
- $\{1, 2, 3\} \supseteq \{1, 2, 3\}$ (every set is a superset of itself).

To show that $A \subseteq B$, we need to prove that if $x \in A$, then $x \in B$. To show that $A \not\subseteq B$, we need to find an element x such that $x \in A$ and $x \notin B$.

Theorem 4.1.1. For any sets A we have $\emptyset \subseteq A$ and $A \subseteq A$.

Proof. To show that $\emptyset \subseteq A$, we need to prove that if $x \in \emptyset$, then $x \in A$. Since there are no elements in the empty set, the statement is vacuously true. Therefore, $\emptyset \subseteq A$ for any set A.
To show that $A \subseteq A$, we need to prove that if $x \in A$, then $x \in A$. This is trivially true since every element of A is also an element of A. Therefore, $A \subseteq A$. □

Definition 4.1.9 (Set Equality).
Two sets A and B are equal, denoted by $A = B$, if they contain exactly the same elements.

This means that $A \subseteq B$ and $B \subseteq A$. We can write this as:

$$A = B \iff \forall x(x \in A \leftrightarrow x \in B)$$

Definition 4.1.10 (Power Set).
The power set of a set A, denoted by $\mathcal{P}(A)$, is the set of all subsets of A. This includes the empty set and A itself. If A has n elements, then the power set $\mathcal{P}(A)$ has 2^n elements.

Example. If $A = \{1, 2\}$, then the power set of A is:

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$$

The size of a set A is denoted by $|A|$. In this case, $|A| = 2$ and $|\mathcal{P}(A)| = 2^2 = 4$.

Note that if $A = \{a, a, b\}$, then $|A| = 2$ because sets do not care about duplicate elements.

4.1.4 Tuples

Definition 4.1.11 (Tuple).
An n-tuple is an ordered collection of n elements, denoted by $(a_1, a_2, \ldots, a_n)$. The order of the elements matters, and duplicate elements are allowed. Two n-tuples are equal if and only if they have the same elements in the same order.

Note that 2-tuples are also called ordered pairs, and 3-tuples are also called ordered triples.

Definition 4.1.12 (Cartesian Product).
The Cartesian product of two sets A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. Formally,

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

Note that $A \times B \neq B \times A$ in general.

Example. If $A = \{1, 2\}$ and $B = \{x, y\}$, then the Cartesian product of A and B is:

$$A \times B = \{(1, x), (1, y), (2, x), (2, y)\}$$

And the product $B \times A$ is:

$$B \times A = \{(x, 1), (x, 2), (y, 1), (y, 2)\}$$

The size of the Cartesian product is given by $|A \times B| = |A| \cdot |B|$. In this case, $|A| = 2$, $|B| = 2$, and $|A \times B| = 2 \cdot 2 = 4$.

Example. The Cartesian product of the sets $A_1, A_2, \ldots, A_n$ is defined as:

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i \text{ for } i = 1, 2, \ldots, n\}$$

is the set of ordered n-tuples. If all sets are equal, i.e., $A_1 = A_2 = \cdots = A_n = A$, then we write:

$$A^n = \underbrace{A \times A \times \cdots \times A}_{n \text{ times}}$$

The size of the Cartesian product in this case is given by $|A^n| = |A|^n$.

Definition 4.1.13 (Truth Set of Predicates).
The truth set of a predicate $P(x)$ is the set of all elements in the domain D of discourse for which the predicate is true. It is denoted by:

$$\{x \in D \mid P(x)\}$$

Example. The truth set of the predicate $P(x)$, where the domain D is the integers and $P(x) = |x| = 1$ is the set $\{-1, 1\}$.

Definition 4.1.14 (Cardinality of a Set).
The cardinality of a set A, denoted by $|A|$, is the number of elements in the set. If A is a finite set, then its cardinality is a non-negative integer. If A is an infinite set, its cardinality is described using concepts from set theory, such as countable and uncountable infinity.

Example. Some examples of cardinality:

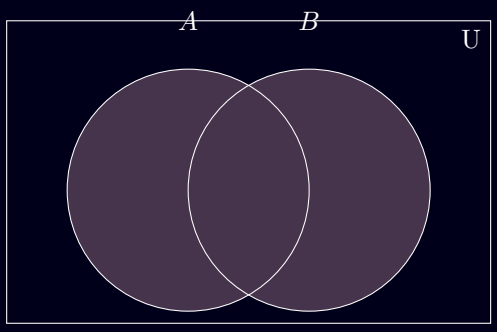
- If $A = \{1, 2, 3\}$, then $|A| = 3$.
- If $B = \{\emptyset\}$, then $|B| = 1$.
- $|\emptyset| = 0$.
- The set of natural numbers $\mathbb{N}$ is infinite and countable, so its cardinality is denoted by $\aleph_0$ (aleph-null).
- The set of real numbers $\mathbb{R}$ is uncountably infinite, and its cardinality is denoted by $2^{\aleph_0}$ (the cardinality of the continuum).

Note that the two last are only added as examples, they will not be covered in this course.

4.1.5 Set Operations

Definition 4.1.15 (Union).
The union of two sets A and B, denoted by $A \cup B$, is the set of all elements that are in A, in B, or in both. Formally,

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$



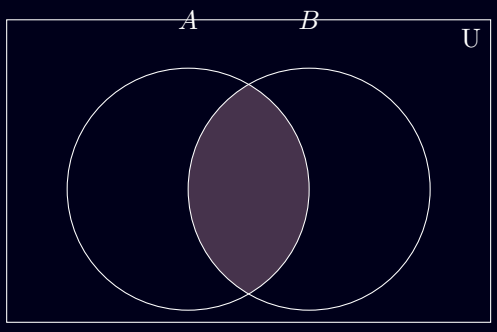
Example. If $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, then the union of A and B is:

$$A \cup B = \{1, 2, 3, 4, 5\}$$

where $5 = |A \cup B| \leq |A| + |B| = 3 + 3 = 6$.

Definition 4.1.16 (Intersection).
The intersection of two sets A and B, denoted by $A \cap B$, is the set of all elements that are in both A and B. Formally,

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$



Example. If $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, then the intersection of A and B is:

$$A \cap B = \{3\}$$

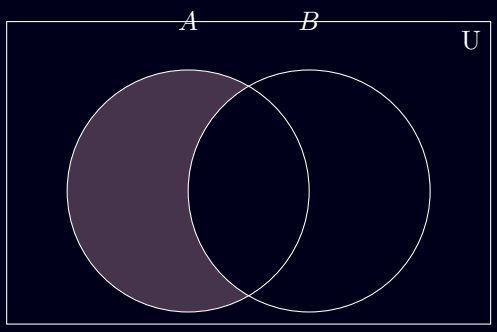
where $|A \cap B| = 1 \leq \min(|A|, |B|) = \min(3, 3) = 3$.

Definition 4.1.17 (Cardinality of Set Union).
The cardinality of the union of two sets A and B is given by the formula:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Definition 4.1.18 (Difference).
The difference of two sets A and B, denoted by $A - B$ or $A \setminus B$, is the set of all elements that are in A but not in B. Formally,

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$



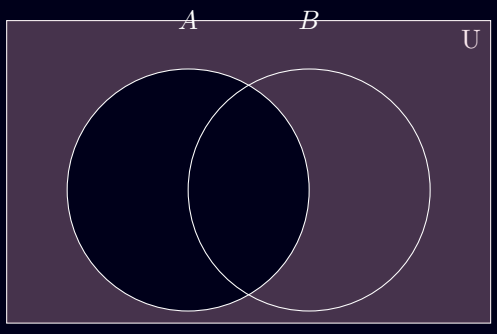
Example. If $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, then the difference of A and B is:

$$A - B = \{1, 2\}$$

where $|A - B| = 2$.

Definition 4.1.19 (Complement).
The complement of a set A with respect to a universal set U, denoted by A^c or $\overline{A}$, is the set of all elements in U that are not in A. Formally,

$$A^c = \{x \in U \mid x \notin A\}$$



Note that the complement of the complement of A is A itself, i.e., $(A^c)^c = A$.

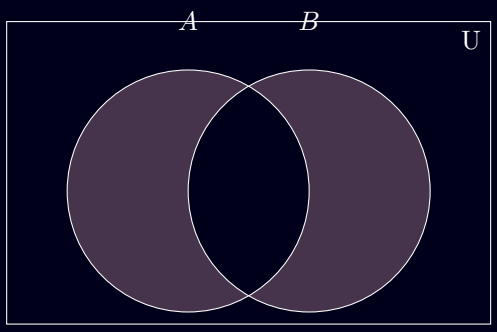
Example. If the universal set $U = \{1, 2, 3, 4, 5\}$ and $A = \{1, 2, 3\}$, then the complement of A is:

$$A^c = \{4, 5\}$$

where $|A^c| = 2$.

Definition 4.1.20 (Symmetric Difference).
The symmetric difference of two sets A and B, denoted by $A \oplus B$ or $A \Delta B$, is the set of elements that are in either A or B but not in both. Formally,

$$A \oplus B = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$$



Example. If $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, then the symmetric difference of A and B is:

$$A \oplus B = \{1, 2, 4, 5\}$$

where $|A \oplus B| = 4$.

4.1.6 Generalised Unions and Intersections

Note that the union and intersection operations are commutative and associative, this means that:

- $A \cup B = B \cup A$ and $A \cap B = B \cap A$ (Commutative Laws).
- $(A \cup B) \cup C = A \cup (B \cup C)$ and $(A \cap B) \cap C = A \cap (B \cap C)$ (Associative Laws).
- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ and $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ (Distributive Laws).

Definition 4.1.21 (Generalised Union and Intersection).
The generalised union and intersection of a collection of sets $\{A_i \mid i \in I\}$, where I is an index set, are defined as follows:

- Generalised Union: $\bigcup_{i \in I} A_i = \{x \mid x \in A_i \text{ for some } i \in I\}$
- Generalised Intersection: $\bigcap_{i \in I} A_i = \{x \mid x \in A_i \text{ for all } i \in I\}$

Example. For $i = 1, 2, \ldots$, let $A_i = \{i, i+1, i+2, \ldots\}$. Then:

$$\bigcup_{i=1}^{n} A_i = A_1 = \{1, 2, 3, \ldots\}$$

and

$$\bigcap_{i=1}^{n} A_i = A_n = \{n, n+1, n+2, \ldots\}$$

4.1.7 Set Operations vs Propositional Calculus Connectives

There is a strong analogy between set operations and propositional calculus connectives:

- Union ($\cup$) corresponds to logical disjunction (OR, $\vee$).
- Intersection ($\cap$) corresponds to logical conjunction (AND, $\wedge$).
- Complement (^{c}) corresponds to logical negation (NOT, $\neg$).
- XOR ($\oplus$) corresponds to logical exclusive disjunction (XOR, $\oplus$).

This strong analogy also extends to the laws governing these operations, such as De Morgan's laws, distributive laws, and associative laws.

Example. De Morgan's laws for sets state that:

- The complement of the union of two sets is the intersection of their complements:

$$(A \cup B)^c = A^c \cap B^c$$

- The complement of the intersection of two sets is the union of their complements:

$$(A \cap B)^c = A^c \cup B^c$$

These laws are analogous to De Morgan's laws in propositional logic:

- The negation of a disjunction is the conjunction of the negations:

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

- The negation of a conjunction is the disjunction of the negations:

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

The list of identities provided in the Logical Equivalences sub-section can be directly translated into set identities by replacing logical connectives with their corresponding set operations.

4.1.8 Proving Set Identities

To prove set identities, we can use various methods such as:

Example. To prove that $(A \cap B)^c = A^c \cup B^c$, we can show that using the Set Builder

Notation:

$$\begin{aligned}
(A \cap B)^c &= \{x \mid x \notin (A \cap B)\} && \text{(Definition of complement)} \\
&= \{x \mid \neg(x \in (A \cap B))\} && \text{(Definition of } \notin\text{)} \\
&= \{x \mid \neg(x \in A \wedge x \in B)\} && \text{(Definition of intersection)} \\
&= \{x \mid x \notin A \vee x \notin B\} && \text{(De Morgan's Law)} \\
&= \{x \mid x \in A^c \vee x \in B^c\} && \text{(Definition of complement)} \\
&= A^c \cup B^c && \text{(Definition of union)}
\end{aligned}$$

Example. Let's prove that a set is a subset of another set, specifically that:

$$A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$$

Thus:

$$\begin{aligned}
A \cap (B \cup C) &= \{x \mid x \in A \wedge x \in (B \cup C)\} && \text{(Definition of intersection)} \\
&= \{x \mid x \in A \wedge (x \in B \vee x \in C)\} && \text{(Definition of union)} \\
&= \{x \mid (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)\} && \text{(Distributive Law)} \\
&= \{x \mid x \in (A \cap B) \vee x \in (A \cap C)\} && \text{(Definition of intersection)} \\
&= (A \cap B) \cup (A \cap C) && \text{(Definition of union)}
\end{aligned}$$

Example. Let's prove that $(A \cap B)^c = A^c \cup B^c$ using a membership table:

$x \in A$	$x \in B$	$x \in A \cap B$	$x \in (A \cap B)^c$	$x \in A^c$	$x \in B^c$	$x \in A^c \cup B^c$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Since the columns for $x \in (A \cap B)^c$ and $x \in A^c \cup B^c$ are identical, we conclude that $(A \cap B)^c = A^c \cup B^c$.

4.2 Functions

Definition 4.2.1 (Function).
Let A and B be two non-empty sets. A function (or mapping, transformation) f from A to B, denoted by $f : A \to B$, is a rule that assigns to every element $a \in A$ exactly one element $b \in B$. This is written as $f(a) = b$.

Definition 4.2.2 (Domain, Codomain, Image, Preimage and Range).
Given a function $f : A \to B$:

- The set A is called the domain of f.

- The set B is called the codomain of f.
- We say that f maps A to B.
- If $f(a) = b$, we say that b is the image of a under f, and a is a preimage of b.
- The range (or image) of f is the set of all images of elements of A, denoted by range(f) or Im(f), and is a subset of B.
- If $S \subseteq A$, then the image of S under f is the set $f(S) = \{f(x) \mid x \in S\} \subseteq B$.

Note that the range is a set of archived values, while the codomain is a set of potential values thus making the range a subset of the codomain.

Example. Let $A = \{a, b, c, d, e\}$ and $B = \{1, 2, 3, 4\}$ with $f(a) = 2$, $f(b) = 1$, $f(c) = 4$, $f(d) = 1$ and $f(e) = 1$.
The image of the subset $S = \{b, c, d\}$ is the set $f(S) = \{1, 4\}$.

Definition 4.2.3 (Real-Valued Function).
A real-valued function is a function whose codomain is the set of real numbers, i.e., $f : A \to \mathbb{R}$.

Definition 4.2.4 (Integer-Valued Function).
An integer-valued function is a function whose codomain is the set of integers, i.e., $f : A \to \mathbb{Z}$.

4.2.1 Adding and Multiplying Functions

Two real-functions or two integer-functions with the same domain can be added and multiplied.

Definition 4.2.5 (Sum and Product of Functions).
Let $f : A \to \mathbb{R}$ and $g : A \to \mathbb{R}$ be two real-valued functions with the same domain A. The sum and product of f and g are defined as follows:

- Sum: $(f + g)(x) = f(x) + g(x)$ for all $x \in A$.
- Product: $(f \cdot g)(x) = f(x) \cdot g(x)$ for all $x \in A$.

Example. Let $f_1(x) = x^2$ and $f_2(x) = x - x^2$ from $\mathbb{R}$ to $\mathbb{R}$. Then:

- The sum of f_1 and f_2 is:

$$(f_1 + f_2)(x) = f_1(x) + f_2(x) = x^2 + (x - x^2) = x$$

- The product of f_1 and f_2 is:

$$(f_1 \cdot f_2)(x) = f_1(x) \cdot f_2(x) = x^2 \cdot (x - x^2) = x^3 - x^4$$

Functions can be represented in various ways, including:

- An explicit statement of the assignment (a graph of nodes and relations).

- A formula (e.g., $f(x) = x^2$).
- A computer program (python program that output n^2 when the input is n).

4.2.2 Injective, Surjective and Bijective Functions

Definition 4.2.6 (Injective Function).
A function $f : A \to B$ is injective (or one-to-one) if different elements in the domain map to different elements in the codomain. Formally, f is injective if:

$$\forall a_1, a_2 \in A, (f(a_1) = f(a_2) \implies a_1 = a_2)$$

Definition 4.2.7 (Surjective Function).
A function $f : A \to B$ is surjective (or onto) if every element in the codomain has at least one preimage in the domain. Formally, f is surjective if:

$$\forall b \in B, \exists a \in A \text{ such that } f(a) = b$$

Definition 4.2.8 (Bijective Function).
A function $f : A \to B$ is bijective if it is both injective and surjective. This means that every element in the codomain is mapped to by exactly one element in the domain.

Example. Let's show that $f(x) = x + 1$ from $\mathbb{Z}$ to $\mathbb{Z}$ is bijective.

- To show that f is injective, assume that $f(a_1) = f(a_2)$ for some $a_1, a_2 \in \mathbb{Z}$. Then:

  $$a_1 + 1 = a_2 + 1 \implies a_1 = a_2$$

  Thus, f is injective.
- To show that f is surjective, let $b \in \mathbb{Z}$ be an arbitrary element in the codomain. We need to find an $a \in \mathbb{Z}$ such that $f(a) = b$. Let $a = b - 1$. Then:

  $$f(a) = f(b - 1) = (b - 1) + 1 = b$$

  Thus, f is surjective.

Since f is both injective and surjective, it is bijective.

In general to show that a function is:

- Injective: Assume $f(a_1) = f(a_2)$ and show that $a_1 = a_2$.
- Not Injective: Find $a_1 \neq a_2$ such that $f(a_1) = f(a_2)$.
- Surjective: Let $b \in B$ and find $a \in A$ such that $f(a) = b$.
- Not Surjective: Find $b \in B$ such that there is no $a \in A$ with $f(a) = b$.

4.2.3 Inverse Functions

Definition 4.2.9 (Inverse Function).
Let $f : A \to B$ be a bijective function. The inverse function of f, denoted by $f^{-1} : B \to A$, is defined by the rule that for each $b \in B$, $f^{-1}(b) = a$ if and only if $f(a) = b$. In other words, f^{-1} reverses the mapping of f.

Example. Let $f(x) = x + 1$ from $\mathbb{Z}$ to $\mathbb{Z}$. We have already shown that f is bijective. To find the inverse function f^{-1}, we need to solve for x in terms of y where $y = f(x)$:

$$y = x + 1 \implies x = y - 1$$

Thus, the inverse function is:

$$f^{-1}(y) = y - 1$$

for all $y \in \mathbb{Z}$.

4.2.4 Composition and Partial Functions

Definition 4.2.10 (Composition of Functions).
Let $f : A \to B$ and $g : B \to C$ be two functions. The composition of f and g, denoted by $g \circ f$, is a function from A to C defined by:

$$(g \circ f)(x) = g(f(x))$$

for all $x \in A$.

Note that the composition of functions is not commutative (i.e., in general, $g \circ f \neq f \circ g$).

Example. Let $f(x) = 2x$ from $\mathbb{R}$ to $\mathbb{R}$ and $g(x) = x + 3$ from $\mathbb{R}$ to $\mathbb{R}$. Then the composition $g \circ f$ is given by:

$$(g \circ f)(x) = g(f(x)) = g(2x) = 2x + 3$$

for all $x \in \mathbb{R}$.
The composition $f \circ g$ is given by:

$$(f \circ g)(x) = f(g(x)) = f(x + 3) = 2(x + 3) = 2x + 6$$

for all $x \in \mathbb{R}$.
Note that $g \circ f \neq f \circ g$.

Definition 4.2.11 (Partial Function).
A partial function f from a set A to a set B is a mapping of each element a in a subset S with $S \subseteq A$ called the domain of definition of f, to a unique element b in B. If $a \in A \setminus S$, we say that f is undefined at a.

When the domain of definition is the entire set A ($S = A$), then f is a total function. A partial function is often used when its exact domain is not known (or difficult to determine) or when it is not necessary to define the function for all elements of A.

Example. Let the function $f : \mathbb{Z} \to \mathbb{N}$ defined as $f(x) = \sqrt{x}$, if and only if, $x = m^2$ (i.e., x is a perfect square for some integer m). Then:

- The domain of definition of f is $S = \{0, 1, 4, 9, 16, \ldots\}$ (the set of all non-negative perfect squares).
- The codomain of f is $\mathbb{N}$ (the set of natural numbers).
- The range of f is $\{0, 1, 2, 3, 4, \ldots\}$ (the set of all natural numbers).
- The function f is undefined for all integers that are not perfect squares (e.g., $f(2)$, $f(3)$, $f(5)$, etc. are undefined).

4.3 Generating Functions

Definition 4.3.1 (Formal Power Sum).
A formal power sum (or formal power series) is an infinite series of the form:

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots = \sum_{n=0}^{\infty} a_n x^n$$

where a_n are coefficients from a given field (e.g., real numbers, complex numbers) and x is an indeterminate. The important part is that this sum is formal, i.e., we are concerned if this infinite sum converges.

To add and multiply formal power sums, we use the following rules:

- Addition: If $F(x) = \sum_{n=0}^{\infty} a_n x^n$ and $G(x) = \sum_{n=0}^{\infty} b_n x^n$, then:

$$F(x) + G(x) = \sum_{n=0}^{\infty} (a_n + b_n) x^n$$

- Multiplication: If $F(x) = \sum_{n=0}^{\infty} a_n x^n$ and $G(x) = \sum_{n=0}^{\infty} b_n x^n$, then:

$$F(x) \cdot G(x) = \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} a_k b_{n-k} \right) x^n$$

These are well defined operations because for a given n, there are only a finite number of operations to perform no matter how large n is. Hence the fact that the sums are infinite does not matter.
To divide formal power sums:

- Division: If $F(x) = \sum_{n=0}^{\infty} a_n x^n$ and $G(x) = \sum_{n=0}^{\infty} b_n x^n$ with $b_0 \neq 0$, then we can find a formal power sum $H(x) = \sum_{n=0}^{\infty} c_n x^n$ such that:

$$F(x) = G(x) \cdot H(x) \quad \Leftrightarrow \quad H(x) = \frac{F(x)}{G(x)}$$

by solving for the coefficients c_n recursively using the relation:

$$c_n = \frac{1}{b_0} \left(a_n - \sum_{k=1}^{n} b_k c_{n-k} \right)$$

for all $n \geq 0$ and $b_0 \neq 0$.

Definition 4.3.2 (Generating Function).
The generating function of a sequence $\{a_n\}_{n=0}^{\infty}$ is the formal power sum:

$$G(x) = \sum_{n=0}^{\infty} a_n x^n$$

where a_n is the n-th term of the sequence. Generating functions are used to study and manipulate sequences in combinatorics and other areas of mathematics.

4.3.1 Solving Recurrence Relations using Generating Functions

Example. Let's solve the reccurence relation $a_n = 3a_{n-1} - 2a_{n-2}$ for $n \geq 2$ with $a_0 = 1$ and $a_1 = 3$ using generating functions. We could solve for a given n using the recurrence relation, but this would be inefficient. Instead, we will use generating functions to find a closed form for a_n.
We start by multiplying both sides of the recurrence relation by x^n and summing over all $n \geq 2$:

$$\sum_{n=2}^{\infty} a_n x^n = \sum_{n=2}^{\infty} 3a_{n-1}x^n - \sum_{n=2}^{\infty} 2a_{n-2}x^n$$

By rewriting the sums to have the same index, we get:

$$\sum_{n=2}^{\infty} a_n x^n = 3x \sum_{n=1}^{\infty} a_n x^n - 2x^2 \sum_{n=0}^{\infty} a_n x^n$$

Let $G(x) = \sum_{n=0}^{\infty} a_n x^n$ be the generating function for the sequence $\{a_n\}$. Then we can rewrite the equation as:

$$G(x) - a_0 - a_1 x = 3x(G(x) - a_0) - 2x^2 G(x)$$

If we group the terms involving $G(x)$ on one side, we get:

$$G(x) - 3xG(x) + 2x^2 G(x) = a_0 + a_1 x - 3xa_0$$

Factoring out $G(x)$, we have:

$$G(x)(1 - 3x + 2x^2) = a_0 + a_1 x - 3xa_0$$

Substituting the initial conditions $a_0 = 1$ and $a_1 = 3$, we get:

$$G(x)(1 - 3x + 2x^2) = 1 + 3x - 3x = 1$$

Since $1 - 3x + 2x^2 \neq 0$, we can solve for $G(x)$:

$$G(x) = \frac{1}{1 - 3x + 2x^2}$$

We can factor the denominator:

$$G(x) = \frac{1}{(1-x)(1-2x)}$$

Using partial fraction decomposition, we can express $G(x)$ as:

$$G(x) = \frac{A}{1-x} + \frac{B}{1-2x}$$

for some constants A and B. Multiplying both sides by the denominator, we get:

$$1 = A(1-2x) + B(1-x)$$

Setting $x = 1$ gives $1 = -A$, so $A = -1$. Setting $x = \frac{1}{2}$ gives $1 = \frac{B}{2}$, so $B = 2$. Thus, we have:

$$G(x) = \frac{-1}{1-x} + \frac{2}{1-2x}$$

We can now use the geometric series formula to expand each term:

$$G(x) = -\sum_{n=0}^{\infty} x^n + 2\sum_{n=0}^{\infty}(2x)^n = -\sum_{n=0}^{\infty} x^n + 2\sum_{n=0}^{\infty} 2^n x^n$$

Combining the sums, we get:

$$G(x) = \sum_{n=0}^{\infty}(-1 + 2^{n+1})x^n$$

Therefore, the closed form for a_n is:

$$a_n = -1 + 2^{n+1} = 2^{n+1} - 1$$

for all $n \geq 0$.

Note that the geometric series formula states that:

$$\frac{1}{1-xr} = 1 + xr + x^2r^2 + x^3r^3 + \ldots = \sum_{n=0}^{\infty}(xr)^n$$

Let's summarise how to solve recurrence relations using generating functions:

- Write down the recurrence relation and initial conditions.
- Multiply both sides of the recurrence relation by x^n and sum over all n.
- Rewrite each expression in terms of the generating function $G(x)$.
- Rearrange terms to get to an expression for $G(x)p(x) = q(x)$.
- Divide by $p(x)$ to get $G(x) = \frac{q(x)}{p(x)}$.
- Factorize $p(x)$ as $p(x) = \prod_{i=1}(1 - r_i x)$, where $\frac{1}{r_i}$ are the roots of the polynomial.
- If all roots are distinct, use partial fraction decomposition to express $G(x)$ as a sum of simpler fractions.
- Determine β_i by solving a linear system of equations.

- Use the geometric series formula to expand each term and find a closed form for a_n.
- Sum up all the terms to get an expression for a_n.

4.3.2 Partial Fraction Expansion

Definition 4.3.3 (Partial Fraction Expansion).
Partial fraction expansion is a method used to decompose a rational function (a fraction where both the numerator and denominator are polynomials) into a sum of simpler fractions.

$$\frac{P(x)}{Q(x)} = \sum_i \frac{A_i}{(x - r_i)^{k_i}} + \sum_j \frac{B_j x + C_j}{(x^2 + px + q)^{m_j}}$$

where $P(x)$ and $Q(x)$ are polynomials, r_i are the roots of the polynomial $Q(x)$, and A_i, B_j, and C_j are constants to be determined.

Example. Let's perform a partial fraction expansion of the rational function:

$$\frac{2x + 3}{(x - 1)(x + 2)}$$

We can express this as:

$$\frac{2x + 3}{(x - 1)(x + 2)} = \frac{A}{x - 1} + \frac{B}{x + 2}$$

for some constants A and B. Multiplying both sides by the denominator $(x-1)(x+2)$, we get:

$$2x + 3 = A(x + 2) + B(x - 1)$$

Expanding the right side, we have:

$$2x + 3 = (A + B)x + (2A - B)$$

Equating coefficients, we get the system of equations:

$$\begin{aligned} A + B &= 2 \\ 2A - B &= 3 \end{aligned}$$

Solving this system, we find $A = 1$ and $B = 1$. Thus, the partial fraction expansion is:

$$\frac{2x + 3}{(x - 1)(x + 2)} = \frac{1}{x - 1} + \frac{1}{x + 2}$$

4.3.3 Derivative of a Formal Power Sum

Definition 4.3.4 (Derivative of a Formal Power Sum).

The derivative of a formal power sum $F(x) = \sum_{n=0}^{\infty} a_n x^n$ is defined as:

$$F'(x) = \sum_{n=1}^{\infty} n a_n x^{n-1}$$

This is obtained by differentiating each term of the power sum with respect to x.

Example. Let $A(x) = 1 + x + x^2 + \ldots$. This serie can be expressed as:

$$A(x) = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

The derivative of $A(x)$ is:

$$A'(x) = \sum_{n=1}^{\infty} n x^{n-1} = \frac{1}{(1-x)^2}$$

Or more explicitly:

$$A'(x) = 1 + 2x + 3x^2 + 4x^3 + \ldots$$

4.4 Exercices

This section gathers a selection of exercises related to Chapter 4, taken from weekly assignments, past exams, textbooks, and other sources. The origin of each exercise will be indicated at its beginning.

Exercise 4.4.1 (*Explique AI*).
How many elements does each of the following sets have where a and b are distinct elements?

(a) $\mathcal{P}(\{a, b, \{a, b\}\})$

(b) $\mathcal{P}(\{\emptyset, a, \{a\}, \{\{a\}\}\})$

(c) $\mathcal{P}(\mathcal{P}(\emptyset))$

Answer:
The number of elements in the power set $\mathcal{P}(S)$ of a set S with n elements is given by 2^n. Therefore:

(a) The set $\{a, b, \{a, b\}\}$ has 3 elements, so its power set has $2^3 = 8$ elements.

(b) The set $\{\emptyset, a, \{a\}, \{\{a\}\}\}$ has 4 elements, so its power set has $2^4 = 16$ elements.

(c) The set $\mathcal{P}(\emptyset)$ is the power set of the empty set, which has 1 element (the empty set itself). Therefore, $\mathcal{P}(\mathcal{P}(\emptyset))$ has $2^1 = 2$ elements.

Chapter 5

Relations and Sequences

5.1 Relations

Definition 5.1.1 (Binary Relations).
A binary relation R from a set A to a set B is a subset of the Cartesian product $A \times B$. In other words, R is a set of ordered pairs (a, b) where $a \in A$ and $b \in B$. We write aRb to indicate that the pair (a, b) is in the relation R.

Example. Let $A = \{0, 1\}$ and $B = \{a, b, c\}$, then:

- $A \times B = \{(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)\}$
- $R_1 = \{(0, a), (1, b)\}$ is a relation from A to B
- $R_2 = \{(0, b), (1, c)\}$ is another relation from A to B

A binary relation R on a set A itself is a subset of $A \times A$. In this case, we say that R is a relation on A.

Example. Let $A = \{0, 1\}$, then:

- $A \times A = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$
- $R_1 = \{(0, 0), (1, 1)\}$ is a relation on A
- $R_2 = \{(0, 1), (1, 0)\}$ is another relation on A

On a set A, there can be many relations. In fact:

- $A \times A$ has $|A|^2$ elements when A has $|A|$ elements.
- Every subset of $A \times A$ can be a relation.
- Therefore, there are $2^{|A|^2}$ possible relations on a set A.

5.1.1 Properties of Relations

Definition 5.1.2 (Reflexive Relation).
A relation R on a set A is called reflexive if for every element $a \in A$, the pair (a, a) is in the relation R. In other words, every element is related to itself.

Note that the empty relation on a empty set is reflexive.

Example. Let $A = \{0, 1\}$, then:

- $R_1 = \{(0, 0), (1, 1)\}$ is reflexive
- $R_2 = \{(0, 1), (1, 0)\}$ is not reflexive
- $R_3 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ is reflexive

Definition 5.1.3 (Symmetric Relation).
A relation R on a set A is called symetric if for every pair $(a, b) \in R$, the pair (b, a) is also in R. In other words, if a is related to b, then b is also related to a.

Example. Let $A = \{0, 1\}$, then:

- $R_1 = \{(0, 0), (1, 1)\}$ is symetric
- $R_2 = \{(0, 1), (1, 0)\}$ is symetric
- $R_3 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ is symetric
- $R_4 = \{(0, 0), (0, 1)\}$ is not symetric

Definition 5.1.4 (Antisymmetric Relation).
A relation R on a set A is called antisymmetric if and only if $(a, b) \in R$ and $(b, a) \in R$ then $a = b,\ \forall a, b \in A$.

Example. Let $A = \{0, 1\}$, then:

- $R_1 = \{(0, 0), (1, 1)\}$ is antisymmetric
- $R_2 = \{(0, 1), (1, 0)\}$ is not antisymmetric
- $R_3 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ is not antisymmetric
- $R_4 = \{(0, 0), (0, 1)\}$ is antisymmetric

Definition 5.1.5 (Transitive Relation).
A relation R on a set A is called transitive if for every pair $(a, b) \in R$ and $(b, c) \in R$, the pair (a, c) is also in R. In other words, if a is related to b and b is related to c, then a is also related to c.

Example. Let $A = \{0, 1, 2\}$, then:

- $R_1 = \{(0, 0), (1, 1), (2, 2)\}$ is transitive

- $R_2 = \{(0,1),(1,0)\}$ is not transitive
- $R_3 = \{(0,0),(0,1),(1,0),(1,1)\}$ is not transitive
- $R_4 = \{(0,0),(0,1),(1,1),(1,2),(0,2)\}$ is transitive
- $R_5 = \{(0,0),(0,1),(1,1),(1,2)\}$ is not transitive

5.1.2 Combining Relations

Definition 5.1.6 (Combining Relations).
Given two relations R_1 and R_2 on a set A, we can combine them using basic operations to form new ones such as:

- Union: $R_1 \cup R_2 = \{(a,b) \mid (a,b) \in R_1 \text{ or } (a,b) \in R_2\}$
- Intersection: $R_1 \cap R_2 = \{(a,b) \mid (a,b) \in R_1 \text{ and } (a,b) \in R_2\}$
- Subtraction: $R_1 - R_2 = \{(a,b) \mid (a,b) \in R_1 \text{ and } (a,b) \notin R_2\}$
- Composition: $R_1 \circ R_2 = \{(a,c) \mid \exists b \in A, (a,b) \in R_1 \text{ and } (b,c) \in R_2\}$

Example. Let $A = \{0,1\}$, $R_1 = \{(0,0),(1,1)\}$ and $R_2 = \{(0,1),(1,0)\}$, then:

- $R_1 \cup R_2 = \{(0,0),(1,1),(0,1),(1,0)\}$
- $R_1 \cap R_2 = \emptyset$
- $R_1 - R_2 = R_1$
- $R_2 - R_1 = R_2$
- $R_1 \circ R_2 = \{(0,1),(1,0)\} = R_2$
- $R_2 \circ R_1 = \{(0,1),(1,0)\} = R_2$

5.1.3 Equivalence Relations and Classes

Definition 5.1.7 (Equivalence Relation).
A relation R on a set A is called an equivalence relation if it is reflexive, symmetric, and transitive.

Two elements a and b that are related by an equivalence relation are called equivalent. The notation $a \sim b$ is often used to denote that a is equivalent to b.

Example. Let $A = \{a,b,c\}$, the relation $R = \{(a,a),(b,b),(c,c),(a,b),(b,a)\}$ is an equivalence relation because:

- Reflexive: $(a,a),(b,b),(c,c) \in R$
- Symmetric: $(a,b) \in R \Rightarrow (b,a) \in R$
- Transitive: There are no pairs (a,b) and (b,c) in R such that (a,c) is not in R

Example. Let $R = \{(a, b) \in \mathbb{R} \times \mathbb{R} \mid a - b \in \mathbb{Z}\}$ be an equivalence relation, then:

- Reflexive: $a - a = 0$, $0 \in \mathbb{Z}$
- Symmetric: $a - b \in \mathbb{Z}$, then $b - a \in \mathbb{Z}$
- Transitive: $a - b \in \mathbb{Z}$ and $b - c \in \mathbb{Z}$, then $(a - b) + (b - c) = a - c \in \mathbb{Z}$

Definition 5.1.8 (Equivalence Class).
Given an equivalence relation R on a set A and an element $a \in A$, the equivalence class of a, denoted by $[a]$, is the set of all elements in A that are equivalent to a under the relation R. In other words:

$$[a] = \{b \in A \mid aRb\}$$

Example. Let $A = \{a, b, c\}$ and $R = \{(a, a), (b, b), (c, c), (a, b), (b, a)\}$ be an equivalence relation on A, then:

- The equivalence class of a is $[a] = \{a, b\}$
- The equivalence class of b is $[b] = \{a, b\}$
- The equivalence class of c is $[c] = \{c\}$

Theorem 5.1.1. Let R be an equivalence relation on a set A. Then the three following statements for element a and b of A are equivalent:

- $(a, b) \in R$
- $[a]_R = [b]_R$
- $[a]_R \cap [b]_R \neq \emptyset$

Proof. Let's prove that these statements are equivalent by showing that (1) implies (2), (2) implies (3), and (3) implies (1).
(1) implies (2): Assume that $(a, b) \in R$. We need to show that $[a]_R = [b]_R$.
Let $c \in [a]_R \subseteq A$. We have:

$$(a, c) \in R \quad \text{and} \quad (a, b) \in R$$

Since R is symmetric we have that $(b, a) \in R$ and thus by transitivity we have:

$$(a, c) \in R \quad \text{and} \quad (b, a) \in R \implies (b, c) \in R$$

Therefore, $c \in [b]_R$. This shows that $[a]_R \subseteq [b]_R$. By a similar argument, we can show that $[b]_R \subseteq [a]_R$. Hence, $[a]_R = [b]_R$.
(2) implies (3): Assume that $[a]_R = [b]_R$. We need to show that $[a]_R \cap [b]_R \neq \emptyset$.
Since R is reflexive, we have $(a, a) \in R$, which implies that $[a]_R$ contains at least a. Therefore, $[a]_R \cap [b]_R$ contains at least a, and hence it is not empty.
(3) implies (1): Assume that $[a]_R \cap [b]_R \neq \emptyset$. We need to show that $(a, b) \in R$.
Since $\exists c \in [a]_R \wedge c \in [b]_R$, we have:

$$(a, c) \in R \quad \text{and} \quad (b, c) \in R$$

Since R is symmetric, we have $(c, b) \in R$. By transitivity, we have:

$$(a, c) \in R \quad \text{and} \quad (c, b) \in R \implies (a, b) \in R$$

This completes the proof that the three statements are equivalent. □

5.1.4 Partitions

Definition 5.1.9 (Partition).
A partition of a set A is a collection of non-empty, disjoint subsets of A whose union is A. It can be formally defined as follows:

- Each subset in the partition is non-empty: $\forall S_i \in P, S_i \neq \emptyset$
- The subsets are pairwise disjoint: $\forall S_i, S_j \in P, i \neq j \implies S_i \cap S_j = \emptyset$
- The union of all subsets in the partition is the entire set A: $\bigcup_{S_i \in P} S_i = A$

Every equivalence relation on a set A induces a partition of A into equivalence classes. Conversely, every partition of a set A defines an equivalence relation on A.

Example. Let $A = \{1, 2, 3, \ldots\}$ and $R = \{(a, b) \mid a - b \text{ is divisible by } 5\} = \{(a, b) \mid a - b = 5k \text{ for some } k \in \mathbb{Z}\}$ be an equivalence relation on A. Then the equivalence classes are:

- $[0]_R = \{0, 5, 10, 15, \ldots\}$
- $[1]_R = \{1, 6, 11, 16, \ldots\}$
- $[2]_R = \{2, 7, 12, 17, \ldots\}$
- $[3]_R = \{3, 8, 13, 18, \ldots\}$
- $[4]_R = \{4, 9, 14, 19, \ldots\}$
- $[5]_R = \{5, 10, 15, 20, \ldots\} \subset [0]_R$

The partition of A induced by the equivalence relation R is:

$$P = \{[1], [2], [3], [4], [0]\}$$

5.1.5 Partial and Total Orders

Definition 5.1.10 (Partial Order).
A relation R on a set A is called a partial order if it is reflexive, antisymmetric, and transitive. A set A equipped with a partial order R is called a partially ordered set or poset, denoted by (A, R).

Example. Let $R = \{(a, b) \in \mathbb{Z} \times \mathbb{Z} \mid a \geq b\}$. We want to show that R is a partial order on $\mathbb{Z}$.

- Reflexive: For any integer a, $a \geq a$
- Antisymmetric: If $a \geq b$ and $b \geq a$, then $a = b$

- Transitive: If $a \geq b$ and $b \geq c$, then $a \geq c$

Therefore, R is a partial order on $\mathbb{Z}$ and $(\mathbb{Z}, \geq)$ is a poset.

Note that the symbol $\preceq$ is used to denote the partial ordering relation in an arbitrary poset.

Example. Let's take the divide relation on $\mathbb{Z}^+$. We show that it is a partial order.

- Reflexive: For any positive integer a, $a \mid a$
- Antisymmetric: If $a \mid b$ and $b \mid a$, then $a = b$
- Transitive: If $a \mid b$ and $b \mid c$, then $a \mid c$

Therefore, the divide relation is a partial order on $\mathbb{Z}^+$ and $(\mathbb{Z}^+, |)$ is a poset.

Definition 5.1.11 (Comparable Elements).
In a poset (A, R), two elements a and b in A are said to be comparable if either aRb or bRa. If neither aRb nor bRa, then a and b are said to be incomparable.

Note that its not necessary that every pair of elements in a poset are comparable.

Example. Consider the poset $(\mathbb{Z}^+, |)$. The elements 2 and 3 are incomparable because neither $2 \mid 3$ nor $3 \mid 2$. However, the elements 2 and 4 are comparable because $2 \mid 4$.

Definition 5.1.12 (Total Order).
A relation R on a set A is called a total order (or linear order) if it is a partial order and every pair of elements in A are comparable. A set A equipped with a total order R is called a totally ordered set or chain, denoted by (A, R).

Example. The poset $(\mathbb{Z}, \leq)$ is a total order because:

- Reflexive: For any integer a, $a \leq a$
- Antisymmetric: If $a \leq b$ and $b \leq a$, then $a = b$
- Transitive: If $a \leq b$ and $b \leq c$, then $a \leq c$
- Comparable: For any integers a and b, either $a \leq b$ or $b \leq a$

Therefore, $(\mathbb{Z}, \leq)$ is a totally ordered set but $(\mathbb{Z}^+, |)$ is not.

Definition 5.1.13 (Well-Order).
A total order R on a set A is called a well-order if every non-empty subset of A has a least element with respect to the order R. In other words, for any non-empty subset $S \subseteq A$, there exists an element $m \in S$ such that for all elements $s \in S$, mRs.

Example. Let's check for the following posets if they are ordered or well-ordered:

- $(\mathbb{Z}, \geq)$ is a total order but not a well-order because the subset of negative integers

does not have a least element.

- $(\mathbb{Z}^+, |)$ is a partial order but not a total order because not every pair of positive integers are comparable.
- $(\mathbb{Z}^+, \leq)$ is a well-order because every non-empty subset of positive integers has a least element.

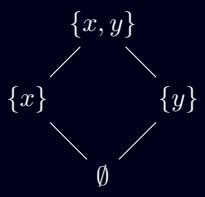
5.1.6 Hasse Diagrams

Definition 5.1.14 (Hasse Diagram).
A Hasse diagram is a graphical representation of a finite partially ordered set (poset). It is a way to visualize the elements of the poset and the order relations between them. In a Hasse diagram:

- Each element of the poset is represented by a vertex (or node).
- An edge (or line) is drawn between two vertices a and b if aRb and there is no element c such that aRc and cRb (i.e., b covers a).
- The edges are drawn such that if aRb, then vertex b is placed higher than vertex a in the diagram.
- Reflexive and transitive relations are not explicitly shown in the diagram.

Example. Let's consider the poset $(\{1, 2, 3, 4\}, \leq)$. The Hasse diagram for this poset is:

Example. Let's draw the Hasse diagram of $(\mathcal{P}(\{x, y\}), \subseteq)$ with $\mathcal{P}(\{x, y\}) = \{\emptyset, \{x\}, \{y\}, \{x, y\}\}$. The Hasse diagram is:

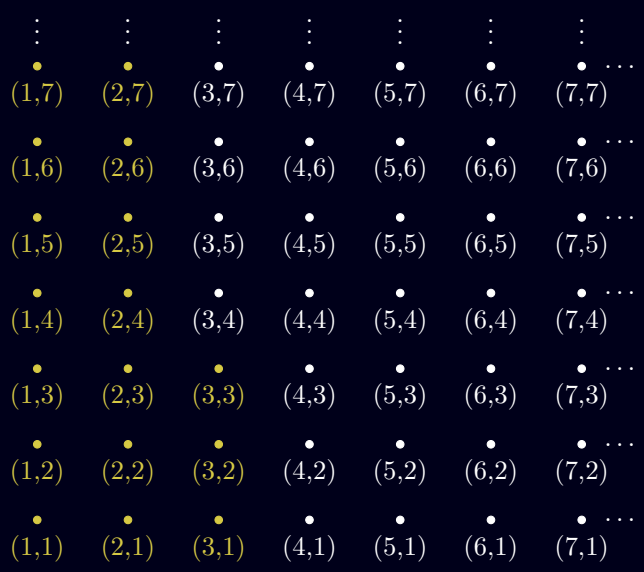


Definition 5.1.15 (Lexicographic Order).
Given two posets (A, R_A) and (B, R_B), the lexicographic order on the Cartesian product $A \times B$ is defined as follows: For any two pairs $(a_1, b_1), (a_2, b_2) \in A \times B$, we say that

$(a_1, b_1) \preceq (a_2, b_2)$ if either:

- $a_1 R_A a_2$, or
- $a_1 = a_2$ and $b_1 R_B b_2$

Example. Let's consider the posets $(\mathbb{Z}^+, \leq)$ and $(\mathbb{Z}^+, \leq)$ then $(\mathbb{Z}^+ \times \mathbb{Z}^+, \prec)$ is a poset with the lexicographic order represented by:



Where all of the colored nodes are the elements that are less than $(3, 4)$.

5.2 Sequences

Definition 5.2.1 (Sequence).
A sequence is an ordered list of elements of a set, which can be finite or infinite. A sequence can be defined as a function from the set of natural numbers (or a subset of natural numbers) to a set of elements.

Example. Some examples of sequences are:

- $1, 2, 3, 4, 5, 8$
- c, o, m, p, u, t, e, r
- $1, 3, 9, 27, 81, \ldots$
- $1, 1, 1, 1, 1, \ldots$

5.2.1 Arithmetic and Geometric Progressions

Definition 5.2.2 (Arithmetic Progression).
An arithmetic progression (or arithmetic sequence) is a sequence of numbers in which the difference between any two consecutive terms is constant. This constant difference is called the common difference, denoted by d. If the first term of the sequence is a_1, then the n-th term of the arithmetic progression can be expressed as:

$$a_n = a_1 + (n-1)d$$

The sum of the first n terms of an arithmetic progression, denoted by S_n, can be calculated using the formula:

$$S_n = \frac{n}{2}(2a_1 + (n-1)d)$$

or equivalently:

$$S_n = \frac{n}{2}(a_1 + a_n)$$

Definition 5.2.3 (Geometric Progression).
A geometric progression (or geometric sequence) is a sequence of numbers in which the ratio between any two consecutive terms is constant. This constant ratio is called the common ratio, denoted by r. If the first term of the sequence is a_1, then the n-th term of the geometric progression can be expressed as:

$$a_n = a_1 \cdot r^{n-1}$$

The sum of the first n terms of a geometric progression, denoted by S_n, can be calculated using the formula:

$$S_n = a_1 \frac{1-r^n}{1-r} \quad \text{if } r \neq 1$$

If $|r| < 1$ and the sequence is infinite, the sum to infinity can be calculated as:

$$S = \frac{a_1}{1-r}$$

Example. Some example of arithmetic and geometric progressions are:

- $1, 4, 9, 16, 25, \ldots$ (not an arithmetic or geometric progression)
- $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ (Geometric Progression)
- $7, 4, 1, -2, -5, \ldots$ (Arithmetic Progression)
- $1, 2, 6, 24, 120, 720$ (not an arithmetic or geometric progression)

5.2.2 Strings

Definition 5.2.4 (Strings).
A string is a finite sequence of characters from a given alphabet. An alphabet is a finite set

of symbols or characters. It can be more formally defined as:

$$f : \{0, 1, \ldots, n\} \to A$$

The empty string, denoted by λ, is the string with zero length.

Example. Let's consider the lowercase English alphabet then the ordering used in dictionaries is a lexicographic order on the set of all strings that can be formed from this alphabet. For example:

- "apple" comes before "banana" because 'a' comes before 'b'
- "discreet" comes before "discrete" because 'e' comes before 't'
- "cat" comes before "catalog" because "cat" is a prefix of "catalog"

Strings with lexicographic ordering are well-ordered sets.

5.2.3 Important Summation Formulas

The following are some important summation and their closed forms that are useful in various mathematical contexts:

- $\sum_{k=0}^{n} ar^k = \frac{ar^{n+1}-a}{r-1} \quad (r \notin \{0, 1\})$
- $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$
- $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad (|x| < 1)$
- $\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2} \quad (|x| < 1)$

Example. If we want to find the formula for $\sum_{k=1}^{n} k$, without using the formula directly. We can do this by using formal power series:

$$A(x) = 1 + x + x^2 + \ldots = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

Now, we can differentiate both sides with respect to x:

$$A'(x) = 0 + 1 + 2x + 3x^2 + \ldots = \sum_{n=1}^{\infty} nx^{n-1} = \frac{1}{(1-x)^2}$$

Multiplying both sides by x gives:

$$xA'(x) = 0 + 1x + 2x^2 + 3x^3 + \ldots = \sum_{n=1}^{\infty} nx^n = \frac{x}{(1-x)^2}$$

Now, if we realize that for any $B(x)$:

$$\begin{aligned} B(x)\frac{1}{1-x} &= (b_0 + b_1x + b_2x^2 + \ldots)(1 + x + x^2 + \ldots) \\ &= b_0 + (b_0 + b_1)x + (b_0 + b_1 + b_2)x^2 + \ldots \\ &= \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} b_k \right) x^n \end{aligned}$$

Then we can take $B(x) = xA'(x)$ and we have:

$$\frac{x}{(1-x)^3} = \sum_{n=0}^{\infty} \left(\sum_{k=1}^{n} k \right) x^n$$

We realize that:

$$\begin{aligned} xA'(x)\frac{1}{1-x} &= \frac{1}{2}xA''(x) = \frac{1}{2}x\left(\sum_{n=0}^{\infty} x^n\right)'' = \frac{1}{2}x\left(\sum_{n=1}^{\infty} nx^{n-1}\right)' \\ &= \frac{1}{2}x\left(\sum_{n=2}^{\infty} n(n-1)x^{n-2}\right) = \frac{1}{2}\left(\sum_{n=2}^{\infty} n(n-1)x^{n-1}\right) \\ &= \sum_{n=1}^{\infty} \left(\frac{n(n+1)}{2}\right) x^n \end{aligned}$$

Therefore, we have:

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

Chapter 6

Algorithms

6.1 Algorithms

Definition 6.1.1 (Algorithm).
An algorithm is a finite sequence of well-defined instructions, typically used to solve a specific task:

- to perform a computation
- to solve a certain problem
- to reach a certain destination

In the context of computing the instructions are carried out by a computer.

Example. Let's find the maximum of a list of numbers:

- Start with the first number as the current maximum.
- Compare the current maximum with the next number in the list.
- If the next number is greater than the current maximum, update the current maximum to this next number.
- Repeat the comparison and update process until all numbers in the list have been checked.
- The current maximum at the end of the process is the maximum number in the list.

Algorithms can be expressed in various ways, including natural language, pseudocode, or programming languages.

Definition 6.1.2 (Pseudocode).
Pseudocode is a high-level description of an algorithm that uses the structural conventions of programming languages but is intended for human reading rather than machine reading. It combines natural language and programming language elements to outline the logic and steps of an algorithm without adhering to the strict syntax of a specific programming language.

Example. Here is a pseudocode example for finding the maximum of a list of numbers:

function MAX_FINDER($a_1, a_2, \ldots, a_n$: integers)
    $max \leftarrow a_1$
    for $i \leftarrow 2$ to n **do**
        if $a_i > max$ **then**
            $max \leftarrow a_i$
        end if
    end for
    return max
end function

Typically algorithms solve problems such as:

- Searching Problems: Finding the position of a particular element in a list
- Sorting Problems: Arranging the elements of a list in a certain order (e.g., ascending or descending)
- Optimization Problems: Determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs

6.1.1 Search Algorithms

Definition 6.1.3 (Search Algorithm).
A search algorithm is a method used to locate a specific item or group of items within a larger collection of data. The goal of a search algorithm is to efficiently find the desired item(s) while minimizing the time and resources required for the search process.

Definition 6.1.4 (Linear Search).
Linear search, also known as sequential search, is a simple search algorithm that checks each element in a list one by one until the desired element is found or the end of the list is reached.

Example. A linear search algorithm would typically look like this:

- First compare x with the first element of the list.
- If they are equal, return the index of the first element.
- If not, compare x with the second element of the list.
- Repeat this process until either x is found or the end of the list is reached.
- If the end of the list is reached without finding x, return -1.

In pseudocode:

function LINEAR_SEARCH($a_1, a_2, \ldots, a_n$: distinct integers, x: integer)
    for $i \leftarrow 1$ to n **do**
        if $a_i = x$ **then**
            return i
        end if

end for
return -1
end function

Example. Consider the list $L = (3, 2, -5, 7, 9, 14, 18)$ and $x = 7$. To find the index of x in L using linear search we will need to check 4 elements: $3, 2, -5, 7$. Let's now compute the average number of comparisons needed to find an element in this list.
Since the list has 7 elements, the average number of comparisons is:

$$\frac{1+2+3+4+5+6+7}{7} = \frac{28}{7} = 4$$

Therefore, on average, we need to check 4 elements to find an element in the list using linear search. More generally, for a list of size n, the average number of comparisons needed to find an element using linear search is:

$$\frac{1+2+3+\ldots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$$

Thus, the average number of comparisons needed to find an element in a list of size n using linear search is $\frac{n+1}{2}$.

Definition 6.1.5 (Binary Search).
Binary search is an efficient algorithm for finding a specific item in a sorted list. It works by repeatedly dividing the search interval in half, eliminating half of the remaining elements from consideration at each step.

Example. A binary search algorithm would typically look like this:

- Start with two pointers, one at the beginning of the list (low) and one at the end of the list (high).
- Calculate the middle index of the current search interval: $mid = \lfloor (low + high)/2 \rfloor$.
- Compare the middle element with the target value x.
- If the middle element is equal to x, return the index of the middle element.
- If x is less than the middle element, update the high pointer to $mid - 1$ to search in the left half of the list.
- If x is greater than the middle element, update the low pointer to $mid + 1$ to search in the right half of the list.
- Repeat steps 2-6 until either x is found or the low pointer exceeds the high pointer (indicating that x is not in the list).
- If x is not found, return -1.

In pseudocode:

function BINARY_SEARCH($a_1, a_2, \ldots, a_n$: distinct integers (sorted), x: integer)
$low \leftarrow 1$

```
    high ← n
    while low < high do
        mid ← ⌊(low + high)/2⌋                    ▷ ⌊·⌋ is rounding down
        if a_mid = x then
            return mid
        else if a_mid ≤ x then
            low ← mid + 1
        else
            high ← mid − 1
        end if
    end while
    return −1
end function
```

Example. Consider the sorted list $L = (2, 3, 5, 7, 9, 14, 18)$ and $x = 5$. To find the index of x in L using binary search we will need to check 3 elements: $7, 3, 9$.

- First iteration: $low = 1$, $high = 7$, $mid = \lfloor (1 + 7)/2 \rfloor = 4$. Compare $a_4 = 7$ with $x = 5$. Since $7 > 5$, we update $high = mid - 1 = 3$.
- Second iteration: $low = 1$, $high = 3$, $mid = \lfloor (1 + 3)/2 \rfloor = 2$. Compare $a_2 = 3$ with $x = 5$. Since $3 < 5$, we update $low = mid + 1 = 3$.
- Third iteration: $low = 3$, $high = 3$, $mid = \lfloor (3 + 3)/2 \rfloor = 3$. Compare $a_3 = 5$ with $x = 5$. They are equal, so we return the index 3.

Therefore, the index of $x = 5$ in the list L is 3.

6.1.2 Sort Algorithms

Definition 6.1.6 (Sorting Algorithm).
A sorting algorithm is a method used to arrange the elements of a list or array in a specific order, typically in ascending or descending order. The goal of a sorting algorithm is to efficiently organize the data while minimizing the time and resources required for the sorting process.

Definition 6.1.7 (Selection Sort).
Selection sort is a simple sorting algorithm that divides the input list into two parts: a sorted sublist and an unsorted sublist. The algorithm repeatedly selects the smallest (or largest, depending on the desired order) element from the unsorted sublist and moves it to the end of the sorted sublist until the entire list is sorted.

Example. A selection sort algorithm would typically look like this:

- Start with the first element of the list as the current minimum.
- Compare the current minimum with the next element in the list.
- If the next element is smaller than the current minimum, update the current minimum

to this next element.

- Repeat the comparison and update process until all elements in the unsorted sublist have been checked.
- Swap the current minimum with the first element of the unsorted sublist.
- Move the boundary between the sorted and unsorted sublists one position to the right.
- Repeat steps 1-6 until the entire list is sorted.

In pseudocode:

function SELECTION_SORT($a_1, a_2, \ldots, a_n$: integers)
  for $i \leftarrow 1$ to $n-1$ **do**
    $min\_index \leftarrow i$
    for $j \leftarrow i+1$ to n **do**
      if $a_j < a_{min\_index}$ **then**
        $min\_index \leftarrow j$
      end if
    end for
    swap(a_i, a_{min_index})
  end for
  return $a_1, a_2, \ldots, a_n$
end function

Example. Consider the list $L = (29, 10, 14, 37, 13)$. To sort this list using selection sort we will need to perform the following steps:

- First pass: Find the minimum element in the entire list, which is 10, and swap it with the first element. The list becomes $(10, 29, 14, 37, 13)$.
- Second pass: Find the minimum element in the remaining unsorted sublist $(29, 14, 37, 13)$, which is 13, and swap it with the second element. The list becomes $(10, 13, 14, 37, 29)$.
- Third pass: Find the minimum element in the remaining unsorted sublist $(14, 37, 29)$, which is 14, and swap it with the third element. The list remains $(10, 13, 14, 37, 29)$.
- Fourth pass: Find the minimum element in the remaining unsorted sublist $(37, 29)$, which is 29, and swap it with the fourth element. The list becomes $(10, 13, 14, 29, 37)$.
- The list is now sorted.

Therefore, after performing selection sort on the list $L = (29, 10, 14, 37, 13)$ we obtain the sorted list $(10, 13, 14, 29, 37)$.

Example. Let's analyze the number of comparisons made by the selection sort algorithm on a list of size n. In the first pass, we compare the first element with the remaining $n-1$ elements, resulting in $n-1$ comparisons. In the second pass, we compare the second element with the remaining $n-2$ elements, resulting in $n-2$ comparisons. This process continues until the second-to-last pass, where we compare the second-to-last element with the last element, resulting in 1 comparison. The total number of comparisons made by the selection

sort algorithm can be calculated as follows:

$$(n-1)+(n-2)+(n-3)+\ldots+2+1=\frac{(n-1)n}{2}=\frac{n^2-n}{2}$$

Therefore, the selection sort algorithm makes a total of $\frac{n^2-n}{2}$ comparisons to sort a list of size n.

Definition 6.1.8 (Bubble Sort).
Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The process is repeated until the list is sorted.

Example. A bubble sort algorithm would typically look like this:

- Start at the beginning of the list.
- Compare the first two adjacent elements.
- If the first element is greater than the second element, swap them.
- Move to the next pair of adjacent elements and repeat steps 2-3 until the end of the list is reached.
- After each pass through the list, the largest unsorted element will have "bubbled up" to its correct position at the end of the list.
- Repeat steps 1-5 for the remaining unsorted portion of the list until no swaps are needed, indicating that the list is sorted.

In pseudocode:

```
function BUBBLE_SORT(a_1, a_2, ..., a_n: integers)
    for i ← 1 to n − 1 do
        for j ← 1 to n − i do
            if a_j > a_{j+1} then
                swap(a_j, a_{j+1})
            end if
        end for
    end for
    return a_1, a_2, ..., a_n
end function
```

Example. Consider the list $L=(5,1,4,2,8)$. To sort this list using bubble sort we will need to perform the following steps:

- First pass: Compare 5 and 1, swap them. List becomes $(1,5,4,2,8)$. Compare 5 and 4, swap them. List becomes $(1,4,5,2,8)$. Compare 5 and 2, swap them. List becomes $(1,4,2,5,8)$. Compare 5 and 8, no swap needed. End of first pass.
- Second pass: Compare 1 and 4, no swap needed. Compare 4 and 2, swap them. List becomes $(1,2,4,5,8)$. Compare 4 and 5, no swap needed. End of second pass.

- Third pass: Compare 1 and 2, no swap needed. Compare 2 and 4, no swap needed. End of third pass.
- The list is now sorted.

Therefore, after performing bubble sort on the list $L = (5, 1, 4, 2, 8)$ we obtain the sorted list $(1, 2, 4, 5, 8)$.

Example. Let's analyze the number of comparisons made by the bubble sort algorithm on a list of size n. In the first pass, we compare the first element with the second element, then the second element with the third element, and so on, until we compare the $(n-1)$-th element with the n-th element. This results in $n-1$ comparisons. In the second pass, we again compare adjacent elements, but this time we only need to compare up to the $(n-2)$-th element, resulting in $n-2$ comparisons. This process continues until the second-to-last pass, where we compare only the first two elements, resulting in 1 comparison. The total number of comparisons made by the bubble sort algorithm can be calculated as follows:

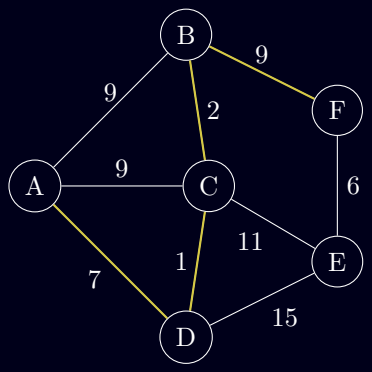
$$(n-1) + (n-2) + (n-3) + \ldots + 2 + 1 = \frac{(n-1)n}{2} = \frac{n^2 - n}{2}$$

Therefore, the bubble sort algorithm makes a total of $\frac{n^2-n}{2}$ comparisons to sort a list of size n.

6.1.3 Greedy Algorithms

Definition 6.1.9 (Greedy Algorithm).
A greedy algorithm is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most immediate benefit or the most optimal choice at that moment. The idea is to make a series of choices, each of which looks best at the time, with the hope that these local optimum choices will lead to a global optimum solution.

Example. Let the graph below represent cities (nodes) connected by roads (edges) with weights representing distances between cities. We want to find the shortest path from city A to city F.



Following the approach of a Greedy algorithm we get:

- From A, the closest city is D (distance 7).
- From D, the closest city is C (distance 1).
- From C, the closest city is B (distance 2).
- From B, the closest city is F (distance 9).

The path taken is A $\rightarrow$ D $\rightarrow$ C $\rightarrow$ B $\rightarrow$ F with a total distance of $7 + 1 + 2 + 9 = 19$. Note that this greedy approach does not always yield the optimal solution for all graphs, like in this case where the optimal path A $\rightarrow$ B $\rightarrow$ F has a total distance of 18.

Example (Cashier's Algorithm). Find for an amount of any n cents the least total number of coins using the following coins: quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent). A greedy algorithm for this problem would look like this:

- Start with the total amount of cents, n.
- While n is greater than 0:
  - If n is greater than or equal to 25, subtract 25 from n and add a quarter to the coin count.
  - Else if n is greater than or equal to 10, subtract 10 from n and add a dime to the coin count.
  - Else if n is greater than or equal to 5, subtract 5 from n and add a nickel to the coin count.
  - Else, subtract 1 from n and add a penny to the coin count.
- Return the total coin count.

In pseudocode:

```
function CASHIER_ALGORITHM(n: integer (amount in cents))
    coin_count ← 0
    while n > 0 do
        if n ≥ 25 then
            n ← n − 25
            coin_count ← coin_count + 1
        else if n ≥ 10 then
            n ← n − 10
            coin_count ← coin_count + 1
        else if n ≥ 5 then
            n ← n − 5
            coin_count ← coin_count + 1
        else
            n ← n − 1
            coin_count ← coin_count + 1
        end if
    end while
    return coin_count
```

end function

For example, if we want to make change for 63 cents, the algorithm would proceed as follows:

- Start with $n = 63$.
- Subtract 25 (quarter), $n = 38$, coin count = 1.
- Subtract 25 (quarter), $n = 13$, coin count = 2.
- Subtract 10 (dime), $n = 3$, coin count = 3.
- Subtract 1 (penny), $n = 2$, coin count = 4.
- Subtract 1 (penny), $n = 1$, coin count = 5.
- Subtract 1 (penny), $n = 0$, coin count = 6.

The total number of coins used is 6 (2 quarters, 1 dime, and 3 pennies).

Theorem 6.1.1. The cashier's algorithm always produces the optimal solution when the coin denominations are 25 cents, 10 cents, 5 cents, and 1 cent (USA).

Proof. Let's prove that the cashier's algorithm always produces the optimal solution for any amount of cents n using the denominations of 25 cents (quarters), 10 cents (dimes), 5 cents (nickels), and 1 cent (pennies). Let q be the number of quarters, d be the number of dimes, n be the number of nickels, and c be the number of cents. Let $(\hat{q}, \hat{d}, \hat{n}, \hat{c})$ be the solution produced by the cashier's algorithm and (q, d, n, c) be the optimal solution. For any amount of cents s, we have:

$$25\hat{q} + 10\hat{d} + 5\hat{n} + \hat{c} = s = 25q + 10d + 5n + c$$

For any solution to be optimal, it must satisfy the following conditions:

- $\hat{c} < 5$ (otherwise we could replace 5 cents with a nickel)
- $\hat{n} < 2$ (otherwise we could replace 2 nickels with a dime)
- $\hat{d} < 3$ (otherwise we could replace 3 dimes with a quarter)

Otherwise we could always reduce the number of coins used by replacing 5 cents with a nickel, 2 nickels with a dime, or 3 dimes with a quarter:

$$(q, d, n, c) \text{ where } n \geq 2 \implies (q, d+1, n-2, c) \text{ a better solution}$$
$$(q, d, n, c) \text{ where } c \geq 5 \implies (q, d, n+1, c-5) \text{ a better solution}$$
$$(q, d, n, c) \text{ where } d \geq 3 \implies (q+1, d-3, n, c) \text{ a better solution}$$

And we know that the greedy solution $(\hat{q}, \hat{d}, \hat{n}, \hat{c})$ satisfies these conditions. We now have:

$$q \leq \hat{q}$$

Let's consider the case where $q = \hat{q} - 1$. Then we have:

$$10d + 5n + c = 25 + 10\hat{d} + 5\hat{n} + \hat{c}$$

Since $c < 5$, $\hat{c} < 5$, $n < 2$, and $\hat{n} < 2$, we must have $d \geq \hat{d} + 3$, which contradicts the optimality of (q, d, n, c). Therefore, we must have $q = \hat{q}$. Similarly, we can show that $d = \hat{d}$, $n = \hat{n}$, and $c = \hat{c}$. Thus, the solution produced by the cashier's algorithm is indeed optimal for any amount of cents n using the denominations of 25 cents, 10 cents, 5 cents, and 1 cent. □

Example. If you apply Cashier's algorithm to make change for 31 cents with only quarters (25 cents), dimes (10 cents), and pennies (1 cent), the algorithm would proceed as follows:

- Start with $n = 31$.
- Subtract 25 (quarter), $n = 6$, coin count $= 1$.
- Subtract 1 (penny) 6 times, $n = 0$, coin count $= 7$.

The total number of coins used is 7 (1 quarter and 6 pennies). However, the optimal solution would be to use 3 dimes and 1 penny, which totals to 4 coins. This example illustrates that Cashier's algorithm does not always yield the optimal solution when the coin denominations changes.

There is no simple formula for assessing, whether for a given set of coin denominations, Cashier's algorithm always produces an optimal solution.

6.1.4 Matching Algorithm

Definition 6.1.10 (Matchings).
Given a finite set A, a matching of A is a set of (unordered) pairs of distinct elements of A where any element occurs in at most one pair (such pairs are called independent).

Definition 6.1.11 (Maximum Matching).
A maximum matching is a matching that contains the largest possible number of independent pairs.

Example. Let $A = \{\text{Lou}, \text{Glenn}, \text{Bobbie}, \text{Jenny}, \text{Tyler}\}$. Then:

- $\{(\text{Lou}, \text{Glenn})\}$ is a matching of A.
- $\{(\text{Lou}, \text{Lou})\}$ is not a matching of A.
- $\{(\text{Lou}, \text{Glenn}), (\text{Lou}, \text{Tyler})\}$ is not a matching of A.
- $\{(\text{Lou}, \text{Glenn}), (\text{Bobbie}, \text{Jenny})\}$ is a matching of A.

Definition 6.1.12 (Preference List).
A preference list L_x defines for every element $x \in A$ the order in which the element prefers to be paired with another element. x prefers y over z if y precedes z on L_x.

Example. Let $A = \{\text{Lou}, \text{Glenn}, \text{Bobbie}, \text{Tyler}\}$ with the following preference lists:

- $L_{\text{Lou}} = [\text{Glenn}, \text{Bobbie}, \text{Tyler}]$
- $L_{\text{Glenn}} = [\text{Bobbie}, \text{Lou}, \text{Tyler}]$
- $L_{\text{Bobbie}} = [\text{Lou}, \text{Glenn}, \text{Tyler}]$
- $L_{\text{Tyler}} = [\text{Lou}, \text{Glenn}, \text{Bobbie}]$

Definition 6.1.13 (Stable Matching).
A matching is stable if there are no two elements x and y that are not paired with each other but prefer each other over their current partners.

Example. Let $A = \{\text{Lou}, \text{Glenn}, \text{Bobbie}, \text{Tyler}\}$ with the following preference lists:

- $L_{\text{Lou}} = [\text{Glenn}, \text{Bobbie}, \text{Tyler}]$
- $L_{\text{Glenn}} = [\text{Bobbie}, \text{Lou}, \text{Tyler}]$
- $L_{\text{Bobbie}} = [\text{Lou}, \text{Glenn}, \text{Tyler}]$
- $L_{\text{Tyler}} = [\text{Lou}, \text{Glenn}, \text{Bobbie}]$

Consider the matchings:

- $M_1 = \{(\text{Lou}, \text{Tyler}), (\text{Glenn}, \text{Bobbie})\}$ is an unstable matching because Lou prefers Bobbie over Tyler, and Bobbie prefers Lou over Glenn.
- $M_2 = \{(\text{Lou}, \text{Bobbie}), (\text{Glenn}, \text{Tyler})\}$ is a stable matching because there are no two elements that prefer each other over their current partners.

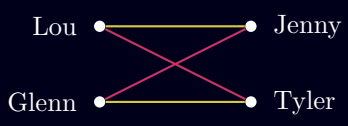
Definition 6.1.14 (Matching Function).
Given a set with even cardinality, partition A into two disjoint subsets A_1 and A_2, with $A_1 \cup A_2 = A$ and $|A_1| = |A_2|$. A matching is a bijection from the elements of one set to the elements of the other set. That means, that pairs can only consist of one element of A_1 and one element of A_2.

Definition 6.1.15 (Marriage Problem).
The marriage problem is to find a stable matching for a given set A with even cardinality and preference lists for each element in A.

Example. Let $A_1 = \{\text{Lou}, \text{Glenn}\}$ and $A_2 = \{\text{Jenny}, \text{Tyler}\}$. Then the preference lists are:

- $L_{\text{Lou}} = [\text{Jenny}, \text{Tyler}]$
- $L_{\text{Glenn}} = [\text{Jenny}, \text{Tyler}]$
- $L_{\text{Jenny}} = [\text{Lou}, \text{Glenn}]$
- $L_{\text{Tyler}} = [\text{Lou}, \text{Glenn}]$

We can make a graph to represent matchings possibility:



The edges in the primary color represent the pairs in the stable matching while the edges in the secondary color represent the pairs not in the stable matching.

Definition 6.1.16 (Gale-Shapley Algorithm).
The Gale-Shapley algorithm is a method for finding a stable matching in the marriage problem. The algorithm works as follows:

- **Proposals:** Each man proposes to women in order of their preferences.
- **Acceptance and Rejection:**
  - If a woman is free, she provisionally accepts the proposal.
  - If she is already engaged, she compares the new proposal with her current engagement.
  - If she prefers the new proposal, she accepts it and rejects her current engagement.
  - Otherwise, she rejects the new proposal.
- **Continue the Process:** Any man who is rejected proposes to the next woman on his list.
- **Termination:** The process continues until every man is engaged.

The algorithm guarantees that the resulting matching is stable, the result is optimal for the side that proposes (in this case man-optimal since men are proposing) and the algorithm is deterministic (always produces the same output for the same input).

Proof. Let's prove that this algorithm produce a stable outcome. Assume for the sake of contradiction that there exists a pair (m, w) and (m', w') in the matching such that m prefers w' over w and w' prefers m over m'. Since m prefers w' over w, he must have proposed to w' before proposing to w. When m proposed to w', there are two possibilities:

- w' was free and accepted m's proposal. In this case, w's current partner is m, which contradicts the assumption that w' prefers m over m'.
- w' was already engaged to m'. Since w' prefers m over m', she would have accepted m's proposal and rejected m'. This contradicts the assumption that m' is w's current partner.

In both cases, we reach a contradiction. Therefore, our initial assumption must be false, and the matching produced by the Gale-Shapley algorithm is stable. □

6.1.5 Unsolvable Problems

Turing proved in 1936 that there are problems that no algorithm can solve. One of the most famous examples of such a problem is the halting problem.

Definition 6.1.17 (Halting Problem).
The halting problem is a decision problem that asks whether a given computer program will eventually halt (stop running) or continue to run indefinitely for a given input.

6.2 Efficiency of Algorithms

Definition 6.2.1 (Asymptotic Analysis).
Asymptotic analysis is a method used to describe the behavior of functions as the input size approaches infinity. It focuses on the dominant terms of a function, which have the most significant impact on its growth rate, while ignoring lower-order terms and constant factors. This allows us to compare the efficiency of algorithms based on their growth rates rather than their exact performance for specific input sizes.

Example. Let $f(n) = 1.5n^2 + 200n + 1000$ be a function to estimate the time to solve a problem of size n. We can decompose $f(n)$ into its dominant term and lower-order terms:

- The constant term: $t(n) = 1000$ which will dominate for small values of n.
- The linear term: $s(n) = 200n$ which will dominate for moderate values of n.
- The quadratic term: $r(n) = 1.5n^2$ which will dominate for large values of n.

As n becomes very large, the dominant term $r(n)$ will have the most significant impact on the growth of $f(n)$. Therefore, we can say that $f(n)$ is asymptotically equivalent to $r(n)$ as n approaches infinity.

6.2.1 Big-O Notation

Definition 6.2.2 (Big-O Notation).
A function $f(n)$ is said to be $O(g(n))$ if there exist positive constants c and n_0 such that for all $n \geq n_0$, the following inequality holds:

$$f(n) \leq c \cdot g(n)$$

It is used to describe an upper bound on the growth rate of a function, indicating that $f(n)$ does not grow faster than a constant multiple of $g(n)$ for sufficiently large values of n.

Example. Let's show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$. We need to find constants c and x_0 such that for all $x \geq x_0$, the inequality $f(x) \leq c \cdot x^2$ holds. We can rewrite the inequality as:

$$x^2 + 2x + 1 \leq c \cdot x^2$$

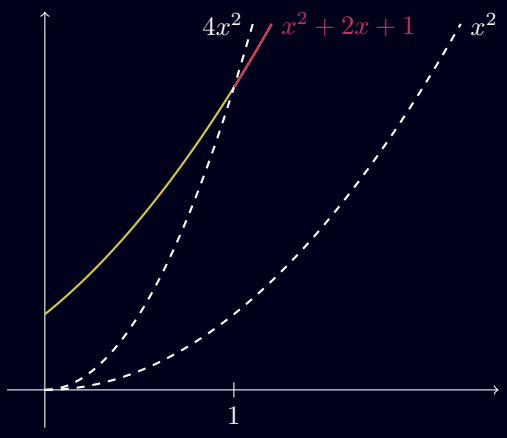
Rearranging the terms, we get:

$$2x + 1 \leq (c - 1) \cdot x^2$$

To satisfy this inequality for sufficiently large x, we can choose $c = 4$ and $x_0 = 1$. For all $x \geq 1$, we have:

$$2x + 1 \leq 3x^2$$

Thus, we have shown that $f(x) = x^2 + 2x + 1$ is $O(x^2)$ with constants $c = 4$ and $x_0 = 1$. We can show to graphically:



where the part of the graph in the secondary color shows that $f(x) < 4x^2$.

Example. Let's determine for the following functions if they are $O(x)$:

- $f(x) = 10$: Yes, with $c = 10$ and $x_0 = 1$.
- $f(x) = 3x + 7$: Yes, with $c = 4$ and $x_0 = 7$.
- $f(x) = x^2 + x + 1$: No, because the quadratic term x^2 grows faster than the linear term x as x approaches infinity.

Remark that for any constant k, $f(n) = k$ is $O(1)$ since we can choose $c = k$ and $n_0 = 1$ but 1 is also $O(k)$ since we can choose $c = 1$ and $n_0 = 1$ (the notation $O(1)$ is generally preferred).

Theorem 6.2.1 (Big-O Estimate for Polynomials)**.** Let $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ be a polynomial of degree k with leading coefficient $a_k \neq 0$. Then, $f(n)$ is $O(n^k)$. In other words, the leading term of the polynomial determines its asymptotic growth rate.

Proof. To prove that $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ is $O(n^k)$, we need to find constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, the following inequality holds:

$$f(n) \leq c \cdot n^k$$

We can rewrite the inequality as:

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \leq c \cdot n^k$$

Rearranging the terms, we get:

$$(c - a_k) n^k - a_{k-1} n^{k-1} - \ldots - a_1 n - a_0 \geq 0$$

To satisfy this inequality for sufficiently large n, we can choose $c = |a_k| + |a_{k-1}| + \ldots + |a_1| + |a_0| + 1$ and $n_0 = 1$. For all $n \geq 1$, we have:

$$(c - a_k) n^k - a_{k-1} n^{k-1} - \ldots - a_1 n - a_0 \geq 0$$

Thus, we have shown that $f(n)$ is $O(n^k)$ with constants $c = |a_k| + |a_{k-1}| + \ldots + |a_1| + |a_0| + 1$ and $n_0 = 1$. □

Theorem 6.2.2 (Sum and Product Rules for Big-O Notation). Let $f(n)$ and $g(n)$ be functions such that $f(n)$ is $O(h(n))$ and $g(n)$ is $O(k(n))$. Then:

- (Sum Rule) The sum $f(n) + g(n)$ is $O(\max(|h(n)|, |k(n)|))$.
- (Product Rule) The product $f(n) \cdot g(n)$ is $O(h(n) \cdot k(n))$.

Proof. **Sum Rule:** Since $f(n)$ is $O(h(n))$, there exist constants $c_1 > 0$ and $n_1 > 0$ such that for all $n \geq n_1$:

$$|f(n)| \leq c_1 \cdot |h(n)|$$

Similarly, since $g(n)$ is $O(k(n))$, there exist constants $c_2 > 0$ and $n_2 > 0$ such that for all $n \geq n_2$:

$$|g(n)| \leq c_2 \cdot |k(n)|$$

Let $n_0 = \max(n_1, n_2)$ and $c = c_1 + c_2$. For all $n \geq n_0$, we have:

$$|f(n) + g(n)| \leq |f(n)| + |g(n)| \leq c_1 \cdot |h(n)| + c_2 \cdot |k(n)|$$

Since $\max(|h(n)|, |k(n)|) \geq |h(n)|$ and $\max(|h(n)|, |k(n)|) \geq |k(n)|$, we can write:

$$|f(n) + g(n)| \leq (c_1 + c_2) \cdot \max(|h(n)|, |k(n)|) = c \cdot \max(|h(n)|, |k(n)|)$$

Thus, $f(n) + g(n)$ is $O(\max(|h(n)|, |k(n)|))$.
Product Rule: Using the same constants as above, for all $n \geq n_0$, we have:

$$|f(n) \cdot g(n)| = |f(n)| \cdot |g(n)| \leq (c_1 \cdot |h(n)|) \cdot (c_2 \cdot |k(n)|) = (c_1 c_2) \cdot |h(n)| \cdot |k(n)|$$

Letting $c' = c_1 c_2$, we have:

$$|f(n) \cdot g(n)| \leq c' \cdot |h(n)| \cdot |k(n)|$$

Thus, $f(n) \cdot g(n)$ is $O(h(n) \cdot k(n))$. □

Remark that if $f_1(n)$ and $f_2(n)$ are both $O(h(n))$, then their sum is also $O(h(n))$.

Theorem 6.2.3 (Transitivity of Big-O Notation). If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.

Proof. Since $f(n)$ is $O(g(n))$, there exist constants $c_1 > 0$ and $n_1 > 0$ such that for all $n \geq n_1$:

$$|f(n)| \leq c_1 \cdot |g(n)|$$

Similarly, since $g(n)$ is $O(h(n))$, there exist constants $c_2 > 0$ and $n_2 > 0$ such that for all $n \geq n_2$:

$$|g(n)| \leq c_2 \cdot |h(n)|$$

Let $n_0 = \max(n_1, n_2)$ and $c = c_1 \cdot c_2$. For all $n \geq n_0$, we have:

$$|f(n)| \leq c_1 \cdot |g(n)| \leq c_1 \cdot (c_2 \cdot |h(n)|) = c \cdot |h(n)|$$

Thus, $f(n)$ is $O(h(n))$. □

Example. Let's determine the Big-O notation of $f(n) = (n^2 + 8) \cdot (n + 1)$. We can use the product rule to evaluate each term of the product separately:

- $n^2 + 8$ is $O(n^2)$ (by the Big-O estimate for polynomials).
- $n + 1$ is $O(n)$ (by the Big-O estimate for polynomials).

Using the product rule, we have:

$$f(n) = (n^2 + 8) \cdot (n + 1) \text{ is } O(n^2) \cdot O(n) = O(n^3)$$

We can also verify this by expanding $f(n)$:

$$f(n) = n^3 + n^2 + 8n + 8$$

By the Big-O estimate for polynomials, we can see that $f(n)$ is indeed $O(n^3)$.

Example. Let's determine the Big-O notation of $f(n) = n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$. We can use the product rule to evaluate each term of the product separately:

- Each term k (where $1 \leq k \leq n$) is $O(n)$.

Using the product rule, we have:

$$\underbrace{O(n) \cdot O(n) \cdot O(n) \cdot \ldots \cdot O(n)}_{n \text{ times}} = O(n^n)$$

Since n is $O(2^n)$ holds (witnesses $k = C = 1$), we have:

$$n \leq 2^n \implies \log_2 n \leq \log_2 2^n = n \cdot \log_2 2 = n$$

Thus, $\log_2 n$ is $O(n)$ (witnesses $k = C = 1$). Furthermore, we have:

$$\frac{\log_2 n}{\log_2 b} = \log_b n \leq \log_2 n \leq n$$

since $\log_2 b$ is a constant for any base $b > 1$. Therefore, $\log_b n$ is also $O(n)$ (witnesses $k = C = 1/\log_2 b$).
Some useful Big-O estimates:

- n^c is $O(n^d)$, but n^d is not $O(n^c)$, for $d > c > 1$
- $(\log_b n)^c$ is $O(n^d)$, but n^d is not $O((\log_b n)^c)$, for $b > 1, c, d > 0$
- n^d is $O(b^n)$, but b^n is not $O(n^d)$, for $b > 1, d > 0$
- b^n is $O(c^n)$, but c^n is not $O(b^n)$, for $c > b > 1$
- c^n is $O(n!)$, but $n!$ is not $O(c^n)$, for $c > 1$

Example. Let's show that if $\lim_{x \to 0} \frac{f(x)}{g(x)} = c$ then $f(x)$ is $O(g(x))$. By the definition of the

limit, if we choose $\epsilon = 1$, we then have:

$$\left|\frac{f(x)}{g(x)} - c\right| \leq 1 \quad \forall x \geq x(\epsilon)$$

Which can be rewritten as:

$$|f(x) - cg(x)| \leq |g(x)|$$

Thus, we have:

$$|f(x)| \leq |cg(x)| + |g(x)| = (|c| + 1) \cdot |g(x)|$$

Therefore, we can choose $n_0 = x(\epsilon)$ and $c' = |c| + 1$ to show that $f(x)$ is $O(g(x))$.

Example. Let's prove that $(\log x)^m$ is $O(x)$ (thus proving that it is also $O(x^c)$ for any $c > 0$). Let's take the limit:

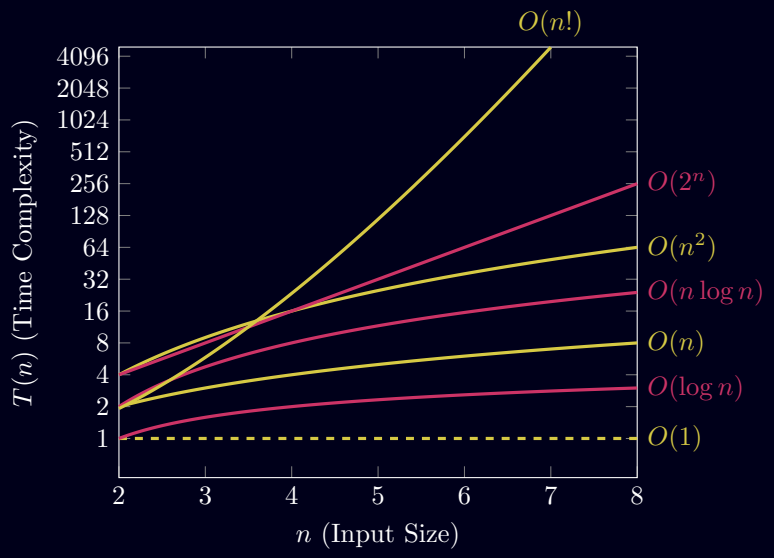
$$\lim_{x \to 0} \frac{(\log x)^m}{x}$$

By using L'Hôpital's rule m times, we get:

$$\lim_{x \to 0} \frac{(\log x)^m}{x} = \lim_{x \to 0} \frac{m(\log x)^{m-1}}{x} = \ldots = \lim_{x \to 0} \frac{m!}{x} = 0$$

Thus, by the previous example, we have shown that $(\log x)^m$ is $O(x)$ (and also little-o $o(x)$).

All of the estimates can be represented in the following graph:



Note that often instead of writing $f(x)$ is $O(g(x))$, it is written $f(x) = O(g(x))$ which is a abuse of the equality sign but widely accepted in computer science. In addition, it can be expressed as $f(x) \in O(g(x))$ which is more precise since $O(g(x))$ is a set of functions.

6.2.2 Big-Omega Notation

Definition 6.2.3 (Big-Omega Notation).
A function $f(n)$ is said to be $\Omega(g(n))$ if there exist positive constants c and n_0 such that for all $n \geq n_0$, the following inequality holds:

$$|f(n)| \geq c \cdot |g(n)|$$

It is used to describe a lower bound on the growth rate of a function, indicating that $f(n)$ does not grow slower than a constant multiple of $g(n)$ for sufficiently large values of n.

Example. Let's show that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$. We need to find constants c and x_0 such that for all $x \geq x_0$, the inequality $f(x) \geq c \cdot x^3$ holds. We can rewrite the inequality as:

$$8x^3 + 5x^2 + 7 \geq c \cdot x^3$$

Rearranging the terms, we get:

$$(8 - c)x^3 + 5x^2 + 7 \geq 0$$

To satisfy this inequality for sufficiently large x, we can choose $c = 7$ and $x_0 = 1$. For all $x \geq 1$, we have:

$$x^3 + 5x^2 + 7 \geq 0$$

Thus, we have shown that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$ with constants $c = 7$ and $x_0 = 1$.

Remark that if $f(n)$ is $O(g(n))$, then $g(n)$ is $\Omega(f(n))$.

Proof. Since $f(n)$ is $O(g(n))$, there exist constants $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, the following inequality holds:

$$|f(n)| \leq c \cdot |g(n)|$$

Rearranging the inequality, we get:

$$|g(n)| \geq \frac{1}{c} \cdot |f(n)|$$

Letting $c' = \frac{1}{c}$, we have:

$$|g(n)| \geq c' \cdot |f(n)|$$

Thus, $g(n)$ is $\Omega(f(n))$. □

6.2.3 Big-Theta Notation

Definition 6.2.4 (Big-Theta Notation).
A function $f(n)$ is said to be $\Theta(g(n))$ if there exist positive constants c_1, c_2, and n_0 such that for all $n \geq n_0$, the following inequalities hold:

$$c_1 \cdot |g(n)| \leq |f(n)| \leq c_2 \cdot |g(n)|$$

It is used to describe a tight bound on the growth rate of a function, indicating that $f(n)$ grows at the same rate as $g(n)$ for sufficiently large values of n.

It is essentially a combination of both Big-O and Big-Omega notations. If $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$, then it is $\Theta(g(n))$.

Example. Let's show that $f(x) = 3x^2 + 8x \log x$ is $\Theta(x^2)$. We need to find constants c_1, c_2, and x_0 such that for all $x \geq x_0$, the following inequalities hold:

$$c_1 \cdot x^2 \leq 3x^2 + 8x \log x \leq c_2 \cdot x^2$$

For the upper bound, we can choose $c_2 = 4$ and $x_0 = 1$. For all $x \geq 1$, we have:

$$3x^2 + 8x \log x \leq 4x^2$$

since $8x \log x$ grows slower than x^2 as x approaches infinity.
For the lower bound, we can choose $c_1 = 3$ and $x_0 = 1$. For all $x \geq 1$, we have:

$$3x^2 + 8x \log x \geq 3x^2$$

Thus, we have shown that $f(x) = 3x^2 + 8x \log x$ is $\Theta(x^2)$ with constants $c_1 = 3$, $c_2 = 4$, and $x_0 = 1$.

Note that if $f(n)$ is $\Theta(g(n))$, then also $g(n)$ is $\Theta(f(n))$.

Theorem 6.2.4 (Estimates for Polynomials). Let $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ be a polynomial of degree k with leading coefficient $a_k \neq 0$. Then, $f(n)$ is $\Theta(n^k)$.

6.2.4 Little-o Notation

Definition 6.2.5 (Little-o Notation).
A function $f(n)$ is said to be $o(g(n))$ if for every positive constant $\varepsilon > 0$, there exists a constant n_0 such that for all $n \geq n_0$, the following inequality holds:

$$|f(n)| < \epsilon \cdot |g(n)| \quad \iff \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

It is used to describe a function that grows strictly slower than another function as the input size approaches infinity.

Example. Let's show that $f(x) = x$ is $o(x^2)$. We need to show that for every positive constant $\varepsilon > 0$, there exists a constant x_0 such that for all $x \geq x_0$, the following inequality holds:

$$|f(x)| < \varepsilon \cdot |x^2|$$

We can rewrite the inequality as:

$$x < \varepsilon \cdot x^2$$

Dividing both sides by x (assuming $x > 0$), we get:

$$1 < \varepsilon \cdot x$$

To satisfy this inequality, we can choose $x_0 = \frac{1}{\varepsilon}$. For all $x \geq x_0$, we have:

$$1 < \varepsilon \cdot x$$

Thus, we have shown that $f(x) = x$ is $o(x^2)$.

6.2.5 Examples of Algorithm Analysis

Note that is this section only the worst-case time complexity is analyzed even if in daily practice the average-case time complexity is often more relevant.

Example. Let's analyze the time complexity of the algorithm to find the maximum element in an array of size n:

```
function MAX_FINDER(a_1, a_2, ..., a_n: integers)
    max ← a_1
    for i ← 2 to n do
        if a_i > max then
            max ← a_i
        end if
    end for
    return max
end function
```

The comparaison operations are colored in primary color. Then, for each element in the array, we have to perform two comparaisons since we started at index 2 (starting index at 1) we only go over $n-1$ elements, so we have to perform $2(n-1)$ comparaisons. We also have to add one comparaison, which is the last one that fails when $i = n+1$. Thus, the total number of comparaisons is:

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

Therefore, the time complexity of the algorithm is $O(n)$.

Example. Let's analyze the time complexity of the Binary Search algorithm to find an element in a sorted array of size $n = 2^k$ where $k = \log n$:

```
function BINARY_SEARCH(a_1, a_2, ..., a_n: distinct integers (sorted), x: integer)
    low ← 1
    high ← n
    while low < high do
        mid ← ⌊(low + high)/2⌋                    ▷ ⌊·⌋ is rounding down
        if a_mid = x then
            return mid
        else if a_mid ≤ x then
            low ← mid + 1
        else
            high ← mid − 1
        end if
    end while
    return −1
end function
```

We clearly see that we use two comparaisons to reduce the list size from 2^k to 2^{k-1}. Thus,

we can write the recurrence relation:

$$T(2^k) = T(2^{k-1}) + 2$$

with the base case $T(1) = 1$ (one comparaison to check if the only element is equal to x). Unrolling the recurrence relation, we get:

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 2 \\ &= T(2^{k-2}) + 2 + 2 \\ &= T(2^{k-3}) + 2 + 2 + 2 \\ &\vdots \\ &= T(2^0) + 2k \\ &= 1 + 2k \end{aligned}$$

Since $k = \log n$, we have:

$$T(n) = 1 + 2\log n$$

Therefore, the time complexity of the algorithm is $O(\log n)$.

In general for the following known agorithms:

- Linear Search: $\Theta(n)$
- Binary Search: $\Theta(\log n)$
- Bubble Sort: $\Theta(n^2)$
- Insertion Sort: $\Theta(n^2)$
- Selection Sort: $\Theta(n^2)$

6.3 P and NP Classes

Definition 6.3.1 (Tractable Problem).
A problem is said to be tractable if there exists an algorithm that can solve the problem in polynomial time, i.e., the time complexity of the algorithm is $O(n^k)$ for some constant k, where n is the size of the input.

If such an algorithm does not exist, the problem is said to be intractable.

Definition 6.3.2 (Class P).
The class P (Polynomial time) is the set of decision problems (problems with a yes/no answer) that can be solved by a deterministic Turing machine in polynomial time. In other words, a problem is in class P if there exists an algorithm that can solve the problem in time $O(n^k)$ for some constant k, where n is the size of the input.

Definition 6.3.3 (Class NP).
The class NP (Nondeterministic Polynomial time) is the set of decision problems for which a given solution can be verified in polynomial time by a deterministic Turing machine. In other words, a problem is in class NP if, given a candidate solution, there exists an algorithm

that can verify whether the solution is correct in time $O(n^k)$ for some constant k, where n is the size of the input.

Example (Knapsack Problem). The Knapsack Problem is a classic example of a problem in class NP. Given a set S of n items, each with a weight w_i and a value v_i, and a maximum weight capacity W, the goal is to determine whether there exists a subset of items from S such that the total weight does not exceed W and the total value is at least a given threshold V.

- No polynomial alogrithm is known
- Exponential time algorithm exists (try all subsets)

If a candidate subset of items is provided, we can verify in polynomial time whether the total weight is less than or equal to W and whether the total value is at least V by summing the weights and values of the items in the subset. Thus, the Knapsack Problem is in class NP.

Note that there exists problems that are not in NP, but they are not commonly encountered in practice.

Chapter 7

Induction and Recursion

Definition 7.0.1 (Induction vs Recursion).
Induction and recursion are different approches to proving results (induction) and solving problems (recursion). They both:

- Rely on the ability to achieve the desired result for the smallest possible version of the problem.
- Induction extends this ability to problems of arbitrary size.
- Recursion breaks down larger problems into smaller subproblems.

7.1 Mathematical Induction

Definition 7.1.1 (Mathematical Induction).
Mathematical induction is a proof technique used to establish the truth of a statement for all natural numbers. It consists of two main steps:

- **Base Case:** Prove that the statement holds for the initial value (usually $n = 0$ or $n = 1$ but can be any natural number).
- **Inductive Step:** Assume the statement holds for some arbitrary natural number k (inductive hypothesis), and then prove that it also holds for $k + 1$.

The definition above can be expressed as rules of inference:

$$(P(1) \land \forall k(P(k) \implies P(k+1))) \implies \forall n P(n)$$

where the domain is the set of positive integers.

Example. Let's prove the following theorem using mathematical induction:

$$n < 2^n \quad \text{for all } n \geq 1$$

Base Case: For $n = 1$:

$$1 < 2^1 \implies 1 < 2 \quad \text{(True)}$$

Inductive Step: Assume the statement holds for some arbitrary $k \geq 1$, i.e., assume:

$$k < 2^k$$

We need to show that it holds for $k + 1$:

$$k + 1 < 2^{k+1}$$

Starting from the inductive hypothesis:

$$k < 2^k$$

Adding 1 to both sides gives:

$$k + 1 < 2^k + 1$$

Since $2^k \geq 2$ for all $k \geq 1$, we have:

$$2^k + 1 \leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$$

Therefore:

$$k + 1 < 2^{k+1}$$

This completes the inductive step. By the principle of mathematical induction, the statement holds for all $n \geq 1$.

7.1.1 Validity of Mathematical Induction

Mathematical induction is valid because of the well-ordering property (previously defined axiom) i.e. every non-empty set of positive integers has a least element.

Proof. Let's make a proof by contradiction to prove the validity of mathematical induction. Assume there exists a property $P(n)$ such that:

- $P(1)$ is true.
- For every $k \geq 1$, if $P(k)$ is true, then $P(k + 1)$ is also true.
- However, there exists some $m \geq 1$ such that $P(m)$ is false.
- Let S be the set of all such m where $P(m)$ is false. By our assumption, S is non-empty.
- By the well-ordering property, S has a least element, say m_0.
- Since m_0 is the least element in S and $m_0 \geq 1$, it follows that $m_0 - 1$ is not in S, which means $P(m_0 - 1)$ is true.
- By the inductive step, since $P(m_0 - 1)$ is true, it follows that $P(m_0)$ must also be true.
- This contradicts our assumption that $P(m_0)$ is false. Therefore, our initial assumption must be incorrect, and there cannot exist such an m where $P(m)$ is false.
- Hence, by mathematical induction, $P(n)$ is true for all $n \geq 1$.

□

Example. Let's prove the formula for the sum of the first n natural numbers using mathematical induction:

$$S(n) = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

Base Case: For $n = 1$:

$$S(1) = 1 = \frac{1(1+1)}{2} = 1 \quad \text{(True)}$$

Inductive Step: Assume the formula holds for some arbitrary $k \geq 1$, i.e., assume:

$$S(k) = \frac{k(k+1)}{2}$$

We need to show that it holds for $k + 1$:

$$S(k+1) = S(k) + (k+1)$$

Using the inductive hypothesis:

$$S(k+1) = \frac{k(k+1)}{2} + (k+1)$$

Simplifying the right-hand side:

$$S(k+1) = \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$$

This completes the inductive step. By the principle of mathematical induction, the formula holds for all $n \geq 1$.

Note that induction is not a method to discover formulas, but rather a method to prove them once they are known.

Example. Let's prove that for any $n \geq 4$ we have:

$$2^n < n!$$

Base Case: For $n = 4$:

$$2^4 = 16 < 24 = 4! \quad \text{(True)}$$

Inductive Step: Assume the statement holds for some arbitrary $k \geq 4$, i.e., assume:

$$2^k < k!$$

We need to show that it holds for $k + 1$:

$$2^{k+1} < (k+1)!$$

Starting from the inductive hypothesis:

$$2^k < k!$$

Multiplying both sides by 2 gives:

$$2^{k+1} < 2 \cdot k!$$

Since $k \geq 4$, we have $2 \cdot k! \leq (k+1) \cdot k! = (k+1)!$. Therefore:

$$2^{k+1} < (k+1)!$$

This completes the inductive step. By the principle of mathematical induction, the statement holds for all $n \geq 4$.

Example. Let's prove the following theorem. If S is a finite set with n elements, where n is a nonnegative integer, then S has 2^n subsets.
Base Case: For $n = 0$, the empty set has exactly one subset (itself), and $2^0 = 1$. Thus, the base case holds.
Inductive Step: Assume the statement holds for some arbitrary $k \geq 0$, i.e., assume that any set with k elements has 2^k subsets. We need to show that it holds for $k + 1$:

$$\text{A set with } k+1 \text{ elements has } 2^{k+1} \text{ subsets.}$$

Let S be a set with $k + 1$ elements. We can write S as:

$$S = T \cup \{a\}$$

where T is a subset of S with k elements and a is an element not in T. By the inductive hypothesis, T has 2^k subsets. Each subset of T can either include the element a or not. Therefore, for each of the 2^k subsets of T, there are two corresponding subsets of S:

- The subset that does not include a (which is just the subset of T).
- The subset that includes a (which is the subset of T plus the element a).

Thus, the total number of subsets of S is:

$$2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$$

This completes the inductive step. By the principle of mathematical induction, the statement holds for all nonnegative integers n.

7.1.2 Strong Induction

Definition 7.1.2 (Strong Induction).
Strong induction is a variation of mathematical induction where the inductive step assumes that the statement holds for all values less than or equal to k, rather than just for k itself. The steps are as follows:

- **Base Case:** Prove that the statement holds for the initial value (usually $n = 0$ or $n = 1$).
- **Inductive Step:** Assume the statement holds for all natural numbers up to k (inductive hypothesis), and then prove that it also holds for $k + 1$.

This can be expressed as rules of inference:

$$(P(1) \land \forall k((P(1) \land P(2) \land \ldots \land P(k)) \implies P(k+1))) \implies \forall n P(n)$$

Note that strong induction is sometimes called the second principle of mathematical induction or complete induction. It is logically equivalent to regular mathematical induction, meaning that any statement that can be proven using one method can also be proven using the other.

Example. Let's prove that every positive integer n can be written as a sum of distinct powers of 2, that is, there exists a set of integers $S = \{k_1, \ldots, k_m\}$ such that:

$$n = \sum_{j=1}^{m} 2^{k_j}$$

Base Case: For $n = 1$:

$$1 = 2^0$$

Thus, the base case holds.
Inductive Step: Assume the statement holds for all positive integers up to k, i.e., assume that for every integer m such that $1 \leq m \leq k$, there exists a set of integers $S_m = \{k_1, \ldots, k_{m'}\}$ such that:

$$m = \sum_{j=1}^{m'} 2^{k_j}$$

We need to show that it holds for $k+1$. There are two cases to consider:

- If $k+1$ is even, then we can write:

  $$k+1 = 2 \cdot \frac{k+1}{2}$$

  By the inductive hypothesis, since $\frac{k+1}{2} \leq k$, there exists a set of integers $S_{\frac{k+1}{2}}$ such that:

  $$\frac{k+1}{2} = \sum_{j=1}^{m'} 2^{k_j}$$

  Therefore:

  $$k+1 = 2 \cdot \sum_{j=1}^{m'} 2^{k_j} = \sum_{j=1}^{m'} 2^{k_j+1}$$

- If $k+1$ is odd, then we can write:

  $$k+1 = 1 + k$$

  By the inductive hypothesis, since $k \leq k$, there exists a set of integers S_k such that:

  $$k = \sum_{j=1}^{m'} 2^{k_j}$$

Therefore:

$$k+1 = 1 + \sum_{j=1}^{m'} 2^{k_j} = 2^0 + \sum_{j=1}^{m'} 2^{k_j}$$

In both cases, we have expressed $k+1$ as a sum of distinct powers of 2. This completes the inductive step. By the principle of strong induction, the statement holds for all positive integers n.

7.1.3 Inductively Defined Sets and Structures

Definition 7.1.3 (Inductively Defined Sets).
An inductively defined set is a set that is defined by specifying a base case (or cases) and one or more rules for generating new elements from existing ones. The process of defining such sets typically involves:

- **Base Case:** Identify one or more initial elements that belong to the set.
- **Inductive Step:** Define rules that allow the construction of new elements from those already in the set.

The set is then the smallest set that contains the base case elements and is closed under the inductive rules.

Remark that only elements generated by the base case and the inductive step belong to the set.

Example. We can defined the set of natural numbers $\mathbb{N}$ inductively as follows:

- **Base Case:** $0 \in \mathbb{N}$
- **Inductive Step:** If $n \in \mathbb{N}$, then $n+1 \in \mathbb{N}$

Thus, the set $\mathbb{N}$ is the smallest set containing 0 and closed under the operation of adding 1.

Example. A subset S of the set of integers inductively defined as follows:

- **Base Case:** $3 \in S$
- **Inductive Step:** If $x \in S$ and $y \in S$, then $x+y \in S$

Thus, the set S contains all integers that can be expressed as sums of the number 3. Therefore, $S = \{3, 6, 9, 12, \ldots\}$, which is the set of all positive multiples of 3.

Definition 7.1.4 (Strings).
A string is a finite sequence of characters from a given alphabet. Formally, if Σ is an alphabet (a finite set of symbols), then a string over Σ is a function $s : \{1, 2, \ldots, n\} \to \Sigma$ for some $n \geq 0$. The set of all strings over Σ is denoted by Σ^*.

Remark that strings can be defined inductively as follows:

- **Base Case:** The empty string λ is in Σ^*.
- **Inductive Step:** If $s \in \Sigma^*$ and $a \in \Sigma$, then the concatenation of s and a, denoted by sa, is also in Σ^*.

Example. Let $\Sigma = \{a, b\}$. Then the strings over Σ include λ, a, b, aa, ab, ba, bb, aaa, and so on. In fact, every finite sequence of a's and b's is a string in Σ^*.

Definition 7.1.5 (Balanced Strings).
A balanced string of parentheses is a string that is defined inductively as follows:

- **Base Case:** The empty string λ is a balanced string.
- **Inductive Step:** If s is a balanced string, then the string (s) is also a balanced string. Additionally, if s_1 and s_2 are balanced strings, then their concatenation $s_1 s_2$ is also a balanced string.

Theorem 7.1.1. Every balanced string has an even number of characters.

Proof. We will prove this theorem using mathematical induction on the length of the balanced string.
Base Case: The empty string λ has 0 characters, which is even. Thus, the base case holds.
Inductive Step: Assume that every balanced string of length k has an even number of characters. We need to show that every balanced string of length $k+1$ also has an even number of characters. There are two cases to consider based on the inductive definition of balanced strings:

- If the balanced string is of the form (s), where s is a balanced string of length $k-1$, then the total length of the string is:

$$\text{Length} = 2 + \text{Length}(s) = 2 + (k-1) = k+1$$

  Since $k-1$ is even by the inductive hypothesis, adding 2 (which is even) results in an even number.

- If the balanced string is of the form $s_1 s_2$, where both s_1 and s_2 are balanced strings with lengths m and n respectively such that $m + n = k + 1$, then by the inductive hypothesis, both m and n are even. The sum of two even numbers is also even:

$$m + n = k + 1$$

In both cases, we have shown that a balanced string of length $k+1$ has an even number of characters. This completes the inductive step. By the principle of mathematical induction, every balanced string has an even number of characters. □

7.2 Recursively Defined Functions

Definition 7.2.1 (Recursively Defined Functions).
A recursively defined function is a function that is defined in terms of itself using base cases and recursive rules. The process of defining such functions typically involves:

- **Base Case:** Specify the value of the function for one or more initial inputs.
- **Recursive Step:** Define the value of the function for other inputs in terms of the

function's values at smaller inputs.

Example. Let f be a function defined recursively as follows:

$$\begin{cases} f(0) = 3 \\ f(n+1) = 2 \cdot f(n) + 3 \quad \text{for } n \geq 0 \end{cases}$$

We can compute the first few values of f:

- $f(0) = 3$
- $f(1) = 2 \cdot f(0) + 3 = 2 \cdot 3 + 3 = 9$
- $f(2) = 2 \cdot f(1) + 3 = 2 \cdot 9 + 3 = 21$
- $f(3) = 2 \cdot f(2) + 3 = 2 \cdot 21 + 3 = 45$

Thus, the first few values of the function can be written as:

$$f(0) = 3 = 1 \cdot 3, \quad f(1) = 9 = 3 \cdot 3, \quad f(2) = 21 = 7 \cdot 3, \quad f(3) = 45 = 15 \cdot 3$$

We can observe that the coefficients $1, 3, 7, 15$ are one less than powers of 2. Specifically:

$$f(n) = (2^{n+1} - 1) \cdot 3$$

We can prove this formula using mathematical induction.
Base Case: For $n = 0$:

$$f(0) = 3 = (2^{0+1} - 1) \cdot 3 = (2^1 - 1) \cdot 3 = 1 \cdot 3 \quad \text{(True)}$$

Inductive Step: Assume the formula holds for some arbitrary $k \geq 0$, i.e., assume:

$$f(k) = (2^{k+1} - 1) \cdot 3$$

We need to show that it holds for $k + 1$:

$$f(k+1) = 2 \cdot f(k) + 3$$

Using the inductive hypothesis:

$$f(k+1) = 2 \cdot ((2^{k+1} - 1) \cdot 3) + 3 = (2^{k+2} - 2) \cdot 3 + 3 = (2^{k+2} - 1) \cdot 3$$

This completes the inductive step. By the principle of mathematical induction, the formula holds for all $n \geq 0$.

Example. Let's define the factorial function $n!$ recursively as follows:

$$\begin{cases} f(0) = 1 \\ f(n+1) = (n+1) \cdot f(n) \quad \text{for } n \geq 0 \end{cases}$$

Example. Let $F(n)$ be the Fibonacci sequence defined recursively as follows:

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(n+2) = F(n+1) + F(n) \quad \text{for } n \geq 0 \end{cases}$$

We want to show that for all $n \geq 3$ we have:

$$F(n) > \alpha^{n-2}$$

where $\alpha = \frac{1+\sqrt{5}}{2}$ is the golden ratio.
Base Cases: For $n = 3$

$$F(3) = F(2) + F(1) = 1 + 1 = 2 > \alpha^{3-2} = \alpha^1 \approx 1.618$$

For $n = 4$:

$$F(4) = F(3) + F(2) = 2 + 1 = 3 > \alpha^{4-2} = \alpha^2 \approx 2.618$$

Thus, the base cases hold.
Inductive Step: Assume the statement holds for all integers up to $k \geq 4$, i.e., assume:

$$F(m) > \alpha^{m-2} \quad \text{for all } 3 \leq m \leq k$$

We need to show that it holds for $k + 1$:

$$F(k+1) = F(k) + F(k-1)$$

Using the inductive hypothesis:

$$F(k+1) > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-3}(\alpha + 1)$$

Since α satisfies the equation $\alpha^2 = \alpha + 1$, we have:

$$F(k+1) > \alpha^{k-3} \cdot \alpha^2 = \alpha^{k-1}$$

This completes the inductive step. By the principle of strong induction, the statement holds for all $n \geq 3$.

Example. Again consider the Fibonacci sequence defined as above. We want to find an exact formula for $F(n)$ using generating functions.
Define the generating function $G(x)$ as follows:

$$G(x) = \sum_{n=0}^{\infty} f_n x^n = f_0 x^0 + f_1 x^1 + \sum_{n=2}^{\infty} f_n x^n = x + \sum_{n=2}^{\infty} f_n x^n$$

where $x \in [0, 1)$ otherwise the series diverges. Using the recursive definition of the Fibonacci

sequence, we can rewrite the summation:

$$G(x) = x + \sum_{n=2}^{\infty}(f_{n-1} + f_{n-2})x^n = x + \sum_{n=2}^{\infty} f_{n-1}x^n + \sum_{n=2}^{\infty} f_{n-2}x^n$$

Re-indexing the summations gives:

$$G(x) = x + x\sum_{n=1}^{\infty} f_n x^n + x^2 \sum_{n=0}^{\infty} f_n x^n = x + xG(x) + x^2 G(x)$$

Rearranging the equation, we have:

$$G(x) - xG(x) - x^2 G(x) = x$$

$$G(x)(1 - x - x^2) = x$$

$$G(x) = \frac{x}{1 - x - x^2}$$

To find an explicit formula for $F(n)$, we can perform partial fraction decomposition on $G(x)$:

$$G(x) = \frac{x}{(1 - \alpha x)(1 - \beta x)}$$

where $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1-\sqrt{5}}{2}$. Thus, we can write:

$$G(x) = \frac{A}{1 - \alpha x} + \frac{B}{1 - \beta x}$$

for some constants A and B. Solving for A and B gives:

$$A = \frac{1}{\sqrt{5}}, \quad B = -\frac{1}{\sqrt{5}}$$

Therefore:

$$G(x) = \frac{1}{\sqrt{5}} \cdot \frac{1}{1 - \alpha x} - \frac{1}{\sqrt{5}} \cdot \frac{1}{1 - \beta x}$$

Using the formula for the sum of a geometric series, we have:

$$G(x) = \frac{1}{\sqrt{5}} \cdot \sum_{n=0}^{\infty} (\alpha x)^n - \frac{1}{\sqrt{5}} \cdot \sum_{n=0}^{\infty} (\beta x)^n$$

$$G(x) = \sum_{n=0}^{\infty} \left(\frac{\alpha^n - \beta^n}{\sqrt{5}} \right) x^n$$

Thus, the explicit formula for the n-th Fibonacci number is:

$$F(n) = \frac{\alpha^n - \beta^n}{\sqrt{5}}$$

Definition 7.2.2 (Recursive Definition of Strings).
The set Σ^* of strings over the alphabet Σ is defined inductively by:

- **Base Case:** $\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).
- **Recursive Step:** If $s \in \Sigma^*$ and $a \in \Sigma$, then the concatenation of s and a, denoted by sa, is also in Σ^*.

Definition 7.2.3 (Length of a String).
The length of a string s, denoted by $l(s)$, is defined recursively as follows:

- **Base Case:** $l(\lambda) = 0$.
- **Recursive Step:** If $s \in \Sigma^*$ and $a \in \Sigma$, then $l(sa) = l(s) + 1$.

Theorem 7.2.1. For any strings $s_1, s_2 \in \Sigma^*$, the length of their concatenation is the sum of their lengths:

$$l(s_1 s_2) = l(s_1) + l(s_2)$$

Proof. We will prove this theorem using mathematical induction on the length of the string s_2.
Base Case: For $s_2 = \lambda$ (the empty string):

$$l(s_1 \lambda) = l(s_1) + l(\lambda) = l(s_1) + 0 = l(s_1)$$

Thus, the base case holds.
Inductive Step: Assume the statement holds for some arbitrary string s_2 of length k, i.e., assume:

$$l(s_1 s_2) = l(s_1) + l(s_2)$$

We need to show that it holds for a string $s_2 a$, where $a \in \Sigma$ and the length of $s_2 a$ is $k + 1$:

$$l(s_1(s_2 a)) = l(s_1 s_2) + 1$$

Using the inductive hypothesis:

$$l(s_1(s_2 a)) = (l(s_1) + l(s_2)) + 1 = l(s_1) + (l(s_2) + 1) = l(s_1) + l(s_2 a)$$

This completes the inductive step. By the principle of mathematical induction, the statement holds for all strings $s_2 \in \Sigma^*$. □

7.3 Recursive Algorithms

Definition 7.3.1 (Recursive Algorithm).
A recursive algorithm is an algorithm that solves a problem by breaking it down into smaller subproblems of the same type. The algorithm typically consists of:

- **Base Case:** A condition under which the algorithm can return a result without further recursion.
- **Recursive Step:** A set of instructions that break the problem into smaller subprob-

lems and call the algorithm recursively on these subproblems.

Note that the difference between recursive algorithms and iterative algorithms is that recursive algorithms rely on function calls to solve subproblems, while iterative algorithms use loops to repeat a set of instructions until a condition is met.

Example. Let's write a recursive algorithm to compute the factorial of a non-negative integer n:

```
function FACTORIAL(n)
    if n = 0 then
        return 1
    else
        return n · FACTORIAL(n − 1)
    end if
end function
```

7.3.1 Divide and Conquer

Definition 7.3.2 (Divide and Conquer).
Divide and conquer is a recursive algorithm design paradigm that involves three main steps:

- **Divide:** Split the problem into smaller subproblems that are similar to the original problem.
- **Conquer:** Solve each subproblem recursively. If the subproblem size is small enough, solve it directly (base case).
- **Combine:** Combine the solutions of the subproblems to form a solution to the original problem.

Example. The algorithm Binary Search, previously defined, is an example of a divide and conquer algorithm. It divides the search interval in half, conquers by recursively searching in the appropriate half, and combines by returning the result of the recursive call. We can write it in pseudocode as follows:

```
function BINARY_SEARCH(A, target, low, high)
    if low > high then
        return −1                                   ▷ Target not found
    end if
    mid ← ⌊(low + high)/2⌋
    if A[mid] = target then
        return mid                                  ▷ Target found
    else if A[mid] < target then
        return BINARY_SEARCH(A, target, mid + 1, high)
    else
        return BINARY_SEARCH(A, target, low, mid − 1)
    end if
end function
```

Let's find the complexity of this algorithm. Let $T(n)$ the worst case complexity of binary

search on an array of size n. In each recursive call, the size of the array is halved. Thus, we can express $T(n)$ as:

$$T(n) \simeq 2T\left(\frac{n}{2}\right) + O(1)$$

where $O(1)$ accounts for the constant time operations performed in each call (like calculating the mid-point and comparing values) and 2 accounts for the two recursive calls. To solve this recurrence relation, we can build a recursion tree:

The height of the tree is $\log_2 n$, and at each level, we perform $O(1)$ work. Therefore, the total work done is:

$$T(n) = O(\log n)$$

Example. Another example of a divide and conquer algorithm is Merge Sort, which sorts an array by recursively dividing it into two halves, sorting each half, and then merging the sorted halves back together. The pseudocode for Merge Sort is as follows:

```
function MERGE_SORT(A)
    if length(A) ≤ 1 then
        return A                          ▷ Base case: array is already sorted
    end if
    mid ← ⌊length(A)/2⌋
    left ← MERGE_SORT(A[0 : mid])
    right ← MERGE_SORT(A[mid : length(A)])
    return MERGE(left, right)
end function
```

The merge function combines two sorted arrays into one sorted array thus it can also be represented by a recursive tree with two branches coming out of each node. The time complexity of Merge Sort can be analyzed using the recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

where $O(n)$ accounts for the time taken to merge the two sorted halves and the 2 accounts for the two recursive calls. Solving this recurrence relation using the Master Theorem gives:

$$T(n) = O(n \log n)$$

Chapter 8

Representations of Numbers

8.1 Integers

Definition 8.1.1 (Decimal Notation).
The decimal notation of an integer $z \in \mathbb{Z}$ is a sequence of digits $d_n d_{n-1} \dots d_1 d_0$ such that

$$z = \sum_{i=0}^{n} d_i \cdot 10^i$$

where each digit d_i is in the set $\{0, 1, 2, \dots, 9\}$.

Example. The decimal notation of the integer 345 is given by the digits $3, 4, 5$:

$$345 = 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$$

Numbers can also be represented in other bases, the most important for computing are: binary (base 2), octal (base 8), and hexadecimal (base 16), but are not limited to these (ancient Mayans used base 20, ancient Babylonians used base 60).

Definition 8.1.2 (Base b Notation).
The base b notation of an integer greater than 1 is a sequence of digits $d_n d_{n-1} \dots d_1 d_0$ such that $0 \leq d_i < b$, n a non negative integer and $d_n \neq 0$.

$$z = \sum_{i=0}^{n} d_i \cdot b^i$$

where each digit d_i is in the set $\{0, 1, 2, \dots, b-1\}$.

Remark that the base b notation is unique for each integer.

Proof. Let $z \in \mathbb{Z}$ be an integer and suppose there exist two different base b notations for z:

$$z = \sum_{i=0}^{n} d_i \cdot b^i = \sum_{i=0}^{m} e_i \cdot b^i$$

where $d_i, e_i \in \{0, 1, 2, \ldots, b-1\}$, $d_n \neq 0$, and $e_m \neq 0$. Without loss of generality, assume $n \geq m$. We can rewrite the equation as:

$$\sum_{i=0}^{n} d_i \cdot b^i - \sum_{i=0}^{m} e_i \cdot b^i = 0$$

This implies:

$$\sum_{i=0}^{m} (d_i - e_i) b^i + \sum_{i=m+1}^{n} d_i b^i = 0$$

Since $d_n \neq 0$, the term $d_n b^n$ is non-zero and dominates the sum for sufficiently large n. Therefore, the only way for the entire sum to equal zero is if all coefficients are zero:

$$d_i - e_i = 0 \quad \text{for } i = 0, 1, \ldots, m$$

and

$$d_i = 0 \quad \text{for } i = m+1, m+2, \ldots, n$$

This leads to a contradiction since $d_n \neq 0$. Hence, our assumption that there exist two different base b notations for z is false. Therefore, the base b notation of an integer is unique. □

Example. Let's represent the integer 17 in common bases:

- Binary (base 2): $17 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 10001_2$
- Octal (base 8): $17 = 2 \cdot 8^1 + 1 \cdot 8^0 = 21_8$
- Hexadecimal (base 16): $17 = 1 \cdot 16^1 + 1 \cdot 16^0 = 11_{16}$

Note that in hexadecimal notation, digits above 9 are represented using letters A to F (A=10, B=11, C=12, D=13, E=14, F=15) so all the numbers from 1 to 15 have a single digit representation.

8.1.1 Contruction of Base b Notation

There are several algorithms to construct the base b notation of an integer $n \geq 0$.

Example. To construct a number n in base b, we can use the repeated subtraction method:

- Find the largest power of b, say b^k, such that $b^k \leq n$.
- Determine the coefficient d_k by calculating $d_k = \lfloor n/b^k \rfloor$.
- Update n to $n - d_k \cdot b^k$.
- Repeat the process for $b^{k-1}, b^{k-2}, \ldots, b^0$ until n becomes zero.

For example, to convert 45 to base 3:

$$3^3 = 27 \leq 45 < 81 = 3^4 \quad \Rightarrow \quad d_3 = \lfloor 45/27 \rfloor = 1$$

$$n = 45 - 1 \cdot 27 = 18$$

$$3^2 = 9 \leq 18 < 27 = 3^3 \quad \Rightarrow \quad d_2 = \lfloor 18/9 \rfloor = 2$$

$$n = 18 - 2 \cdot 9 = 0$$

$$3^1 = 3 \leq 0 < 9 = 3^2 \quad \Rightarrow \quad d_1 = \lfloor 0/3 \rfloor = 0$$

$$3^0 = 1 \leq 0 < 3 = 3^1 \quad \Rightarrow \quad d_0 = \lfloor 0/1 \rfloor = 0$$

Thus, reading the coefficients from d_3 to d_0, we get $45 = 1200_3$.

Theorem 8.1.1 (Division Remainder Method)**.** If a is an integer and d a positive integer, then there are unique integers q and r such that $0 \leq r < d$ and:

$$a = d \cdot q + r$$

The integer q is called the quotient, d is called the divisor, a is called the dividend and r the remainder of the division of a by d.

Example. For example, dividing 17 by 5:

$$17 = 5 \cdot 3 + 2$$

Here, the quotient $q = 3$ and the remainder $r = 2$.

Example. Another example, dividing -11 by 3:

$$-11 = 3 \cdot (-4) + 1$$

Here, the quotient $q = -4$ and the remainder $r = 1$.

Example. To construct a number n in base b, we can use the division-remainder method:

- Divide n by b to get a quotient q_0 and a remainder d_0 (the least significant digit).
- Set $n = q_0$ and repeat the division until the quotient is zero.
- The base b representation is obtained by reading the remainders in reverse order.

For example, to convert 45 to base 2:

$$45 \div 2 = 22 \text{ remainder } 1 \quad (d_0 = 1)$$

$$22 \div 2 = 11 \text{ remainder } 0 \quad (d_1 = 0)$$

$$11 \div 2 = 5 \text{ remainder } 1 \quad (d_2 = 1)$$

$$5 \div 2 = 2 \text{ remainder } 1 \quad (d_3 = 1)$$

$$2 \div 2 = 1 \text{ remainder } 0 \quad (d_4 = 0)$$

$$1 \div 2 = 0 \text{ remainder } 1 \quad (d_5 = 1)$$

Reading the remainders in reverse order, we get $45 = 101101_2$.

8.1.2 Operations on Base b Notation

Definition 8.1.3 (Addition in Base b).
To add two numbers in base b, align the digits and add them column by column from right to left, carrying over any value that exceeds $b-1$ to the next column.

Example. For example, adding 345_8 and 267_8 in base 8:

$$\begin{array}{r} 345_8 \\ +267_8 \\ \hline 634_8 \end{array}$$

Here, $5+7=12_{10}=14_8$ (write down 4, carry over 1), $4+6+1=11_{10}=13_8$ (write down 3, carry over 1), and $3+2+1=6_{10}=6_8$.

Definition 8.1.4 (Multiplication in Base b).
To multiply two numbers in base b, use the standard multiplication algorithm, multiplying each digit of the second number by the entire first number, shifting left for each digit position, and then summing all the partial products. The pseudo-code is as follows:

Remark that mutiplying a number by the base b is equivalent to shifting the number one position to the left (adding a zero at the end).

Example. For example, multiplying 23_5 and 14_5 in base 5:

$$\begin{array}{r} 23_5 \\ \times\ 14_5 \\ \hline 202_5 \\ +230_5 \\ \hline 432_5 \end{array}$$

Here, $3\times 4=12_{10}=22_5$ (write down 2, carry 2), then $2\times 4+2=10_{10}=20_5$ (write down 0, carry 2), so the partial product is 202_5; the other partial product is 23_5 shifted to 230_5; adding these gives $202_5+230_5=432_5$.

8.2 Counting

Definition 8.2.1 (Counting).
Counting is ubiquitous in mathematics (e.g., combinatorics) and computer science. It involves determining the number of elements in a set or the number of ways to arrange or select items. Various techniques such as permutations, combinations, and the principle of inclusion-exclusion are used in counting problems.

Let's introduce some notations that will be useful in counting:

- Sequences are ordered.
- X will be refered to as the alphabet.

- Often $s(1), s(2), \ldots, s(n)$ will be denoted as $s = s_1, s_2, \ldots, s_n$.
- Sequences will be used interchangeably with strings/words.

Example. Given the set of vowels $X = \{a, e, i, o, u\}$, the number of possible 4-letter sequences (words) that can be formed is:

$$|X|^4 = 5^4 = 625$$

since each position in the sequence can be filled by any of the 5 vowels.

Theorem 8.2.1 (Product Rule). If a task can be broken down into k sequential steps, where the first step can be performed in n_1 ways, the second step in n_2 ways, and so on up to the k-th step which can be performed in n_k ways, then the total number of ways to perform the entire task is given by:

$$N = n_1 \times n_2 \times \ldots \times n_k = \prod_{i=1}^{k} n_i$$

Remark that the set from which we choose the elements can change at each step but the set must not depend on the previous choices.

Example. If a license plate consists of 2 letters followed by 3 digits, and there are 26 letters in the alphabet and 10 digits (0-9), the total number of different license plates that can be formed is:

$$N = 26^2 \times 10^3 = 676000$$

Theorem 8.2.2. The number of different subset of a set S with n elements is 2^n.

Proof. When the element of S are listed in an arbitrary order, there is a one-to-one correspondence between the subsets of S and the binary sequences of length n: the i-th element of S is in the subset if and only if the i-th digit of the sequence is 1. Since there are 2^n binary sequences of length n, there are 2^n subsets of S. □

Remark that if the cardinality of a set is known and a bijection can be established between this set and another set, then the cardinality of the second set is also known and it is the same as the first set.

8.2.1 Counting Functions

Definition 8.2.2 (Counting Functions).
Given two finite sets A and B with cardinalities $|A| = m$ and $|B| = n$, the number of functions from A to B is given by:

$$n^m$$

since each element in A can be mapped to any of the n elements in B.

Remark that if instead of functions, only injective (one-to-one) functions are considered, the counting changes to:

$$\frac{n!}{(n-m)!}$$

provided that $n \geq m$.

Example. Rob has 4 blue socks, 7 red socks, 5 white socks and 3 black socks. He likes to wear either a red sock on his left foot with a blue sock on his right foot or a white sock on his left foot with a black sock on his right foot. How many different ways can Rob wear his socks?

$$\text{Total ways} = (7 \times 4) + (5 \times 3) = 28 + 15 = 43$$

Theorem 8.2.3 (Sum Rule). If a task can be performed in n_1 ways or n_2 ways (but not both), then the total number of ways to perform the task is:

$$N = n_1 + n_2$$

Example. Each user on a computer system has a password, which is 6 to 8 characters long, where each character is an uppercase letter (A-Z) or a digit (0-9). How many different passwords are possible?

$$\text{Total passwords} = 36^6 + 36^7 + 36^8 = 2,901,650,853,888$$

Example. Same example as before but now the password must contain at least one digit. How many different passwords are possible?

$$\text{Total passwords} = (36^6 - 26^6) + (36^7 - 26^7) + (36^8 - 26^8) = 2,743,303,001,088$$

Theorem 8.2.4 (Substraction Rule). If a task can be performed in n ways, and m of these ways are not allowed, then the total number of ways to perform the task is:

$$N = n - m$$

Example. How many bit strings of length 8 either start with a 1 bit or end with the two bits 00?

$$\text{Total bit strings} = 2^7 + 2^6 - 2^5 = 160$$

Note that the bit strings that both start with a 1 and end with 00 have been subtracted once to avoid double counting.
We could also view this example has two sets: A the set of bit strings of length 8 that start with a 1 and B the set of bit strings of length 8 that end with 00. The cardinality of the union of these two sets is given by:

$$|A \cup B| = |A| + |B| - |A \cap B| = 2^7 + 2^6 - 2^5 = 160$$

Example. How many integers from 1 to 100 are not divisible by 2 or 5?

$$\text{Total integers} = 100 - (50 + 20 - 10) = 40$$

Here, 50 integers are divisible by 2, 20 integers are divisible by 5 and 10 integers are divisible by both 2 and 5.

8.3 Permutations and Combinations

Let's assume that $S = \{1, 2, 3, 4\}$. The number of ways to build a strings of length 2 from the elements of S:

	Permutation	**Combination**
without repetition	12 13 14 21 23 24 31 32 34 41 42 43	12 13 14 23 24 34
with repetition	11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44	11 12 13 14 22 23 24 33 34 44

8.3.1 Permutations

Definition 8.3.1 (Permutations).
A permutation of a set of n distinct elements is an arrangement of all the elements in a specific order. The number of different permutations of n distinct elements is given by:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

Remark that if only r elements are to be arranged from a set of n distinct elements, the number of different permutations is given by:

$$P(n, r) = \frac{n!}{(n-r)!}$$

Example. Let's take two similar examples:

- A class of 100 sutdents is electing a president, a vice-president and a secretary. How many different ways can these positions be filled?

$$P(100, 3) = \frac{100!}{(100-3)!} = 970200$$

- A class of 100 is electing 3 representatives. How many different ways can these positions be filled? Let's denote the representatives as R_1, R_2 and R_3 to distinguish them, then the number of different ways to fill these positions is:

$$\begin{matrix} R_1 & R_2 & R_3 \\ R_1 & R_3 & R_2 \\ R_2 & R_1 & R_3 \\ & \vdots & \\ R_3 & R_2 & R_1 \end{matrix}$$

There are $3! = 6$ ways to arrange the representatives for each selection of 3 students. Therefore, the total number of different ways to select the representatives is:

$$C(n,k) = \frac{P(n,k)}{k!} = \binom{n}{k} = \frac{P(100,3)}{3!} = 161700$$

8.3.2 Combinations

Definition 8.3.2 (Binomial Coefficient).
The binomial coefficient is defined as:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Remark that this is the same coefficient used to expand the binomial expression $(x+y)^n$ using the Binomial Theorem.

Example. Let's expend the binomial expression $(x+y)^4$ using the Binomial Theorem:

$$(x+y)^4 = \sum_{k=0}^{4} \binom{4}{k} x^{4-k}y^k = \binom{4}{0}x^4 + \binom{4}{1}x^3y + \binom{4}{2}x^2y^2 + \binom{4}{3}xy^3 + \binom{4}{4}y^4$$

Calculating the binomial coefficients, we get:

$$(x+y)^4 = 1 \cdot x^4 + 4 \cdot x^3y + 6 \cdot x^2y^2 + 4 \cdot xy^3 + 1 \cdot y^4$$

Theorem 8.3.1 (Combinations). The number of ways to choose k elements from a set of n distinct elements, where the order of selection does not matter, is given by the binomial coefficient:

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Remark that $C(n,k) = C(n,n-k)$ since choosing k elements to include is equivalent to choosing $n-k$ elements to exclude.

Example. How many poker hands of 5 cards can be dealt from a standard deck of 52 cards?

$$C(52,5) = \binom{52}{5} = \frac{52!}{5!(52-5)!} = 2,598,960$$

Example. Based on the previous example, how many poker hands with a full house (can be three of a kind and a pair) can be dealt from a standard deck of 52 cards?

$$\text{Total full house hands} = 13 \times C(4,3) \times 12 \times C(4,2) = 3,744$$

Here, 13 is the number of ranks for the three of a kind, $C(4,3)$ is the number of ways to choose 3 suits from 4, 12 is the number of remaining ranks for the pair, and $C(4,2)$ is the number of ways to choose 2 suits from 4.

Example. A fruit shop offers 5 types of fruits: apples, bananas, cherries, dates, and elderberries. A customer wants to buy a fruit basket containing 8 pieces of fruit, where the order of selection does not matter and repetitions are allowed. How many different fruit baskets can the customer create?
This problem can be solved using the "Stars and Bars" method. We represent the $r = 8$ fruits as stars ($*$) and use $n-1$ bars ($|$) to separate the $n = 5$ different types of fruit. We need $5-1 = 4$ bars to create 5 compartments (bins).
For example, if we select 2 apples, 3 bananas, 1 cherry, 0 dates, and 2 elderberries, this corresponds to the string:

$$**\,|\,***\,|\,*\,||\,**$$

Any arrangement of these 8 stars and 4 bars represents a unique fruit basket. The total length of the string is $8+4 = 12$. The problem reduces to choosing which 8 of the 12 positions contain stars (or equivalently, which 4 positions contain bars).
The formula for the number of combinations with repetition is thus given by:

$$C(n+r-1,r) = \binom{n+r-1}{r}$$

In this case, $n = 5$ and $r = 8$:

$$\text{Total fruit baskets} = \binom{5+8-1}{8} = \binom{12}{8} = \frac{12!}{8!4!} = 495$$

Therefore, the customer can create 495 different fruit baskets.

8.3.3 Permutations with Indistinguishable Objects

Theorem 8.3.2 (Permutations with Indistinguishable Objects)**.** The number of distinct permutations of n objects, where there are n_1 indistinguishable objects of type 1, n_2 indistinguishable objects of type 2, ..., and n_k indistinguishable objects of type k (with

$n_1 + n_2 + ... + n_k = n$), is given by:

$$\frac{n!}{n_1! \times n_2! \times ... \times n_k!}$$

Proof. Consider a set of n objects, where n_1 are of type 1, n_2 are of type 2, ..., and n_k are of type k. The total number of permutations of these n objects, if they were all distinguishable, would be $n!$. However, since the objects of the same type are indistinguishable, we need to account for the overcounting that occurs when we permute the indistinguishable objects among themselves. For each type i, there are $n_i!$ ways to arrange the n_i indistinguishable objects of that type. Therefore, to find the number of distinct permutations, we divide the total permutations $n!$ by the product of the factorials of the counts of each type:

$$\text{Distinct permutations} = \frac{n!}{n_1! \times n_2! \times ... \times n_k!}$$

This formula gives us the correct count of distinct arrangements by eliminating the overcounting due to indistinguishable objects. □

Remark that this theorem could also be proved using the product rule by considering the selection of positions for each type of indistinguishable object sequentially ($C(n, n_1) \cdot C(n - n_1, n_2) \cdot \ldots \cdot C(n - n_1 - ... - n_{k-1}, n_k)$). which would lead to the same result.

Example. How many different strings can be made by reordering the letters of the word "SUCCESS"?

$$\text{Total strings} = \frac{7!}{3!2!1!1!} = 420$$

Here, the total number of letters is 7, with the letter 'S' appearing 3 times, 'C' appearing 2 times, and 'U' and 'E' appearing 1 time each. The formula accounts for the indistinguishable letters by dividing by the factorial of their counts.

8.3.4 Repetition vs. Indistinguishable Objects

Example. Let's suppose we have the following alphabet $S = \{s, u, c\}$ and we build strings of length 4 from the elements of S.
If repetition is allowed, the total number of strings is:

$$|S|^4 = 3^4 = 81$$

If repetition are not allowed, then the number of strings with the alphabet $S' = \{s, s, u, c\}$ is:

$$\text{Total strings} = \frac{4!}{2!1!1!} = 12$$

Here, the total number of letters is 4, with the letter 's' appearing 2 times, and 'u' and 'c' appearing 1 time each. The formula accounts for the indistinguishable letters by dividing by the factorial of their counts.

8.4 Pigeonhole Principle

Theorem 8.4.1 (Pigeonhole Principle). If n items are put into m containers, with $n > m$, then at least one container must contain more than one item.

Proof. Assume, for the sake of contradiction, that no container contains more than one item. This means that each container can hold at most one item. Therefore, the maximum number of items that can be placed in m containers is m. However, since $n > m$, this leads to a contradiction because we have more items than the maximum capacity of the containers. Hence, our assumption is false, and at least one container must contain more than one item. □

Theorem 8.4.2 (Generalized Pigeonhole Principle). If n items are put into m containers, then at least one container must contain at least $\lceil n/m \rceil$ items.

Proof. Assume, for the sake of contradiction, that no container contains $\lceil n/m \rceil$ or more items. This means that each container can hold at most $\lceil n/m \rceil - 1$ items. Therefore, the maximum number of items that can be placed in m containers is:

$$m \times (\lceil n/m \rceil - 1)$$

Since $\lceil n/m \rceil - 1 < n/m$, we have:

$$m \times (\lceil n/m \rceil - 1) < m \times (n/m) = n$$

This leads to a contradiction because we have more items than the maximum capacity of the containers. Hence, our assumption is false, and at least one container must contain at least $\lceil n/m \rceil$ items. □

Example. Let's show that among 100 students, at least one month has at least 9 students born in that month.
There are 12 months in a year, so we have $n = 100$ students and $m = 12$ months. Using the generalized pigeonhole principle:

$$\text{At least one month has } \lceil 100/12 \rceil = 9 \text{ students.}$$

8.4.1 Minimum Number of Items to Guarantee a Certain Outcome

Definition 8.4.1 (Minimum Number of Items to Guarantee a Certain Outcome).
To guarantee that at least k items are in the same container when n items are distributed among m containers, we need at least:

$$n = m \times (k - 1) + 1$$

items.

Example. What is the minimum number of students required in a physics class to guarantee that at least 6 students receive the same grade, assuming the possible grades are A, B, C, D, and F?
There are 5 possible grades, so we have $m = 5$ grades. To ensure that at least one grade

has at least 6 students, we can use the generalized pigeonhole principle:

$$n = m \times (k-1) + 1 = 5 \times (6-1) + 1 = 26$$

Therefore, a minimum of 26 students is required to guarantee that at least 6 students receive the same grade.

Example. How many numbers must be selected from the set $\{1, 2, 3, 4, 5, 6\}$ to guarantee that at least one pair of these numbers add up to 7?
The pairs that add up to 7 are: $(1, 6)$, $(2, 5)$, and $(3, 4)$. Thus, we have $m = 3$ pairs. To ensure that at least one pair is selected, we can use the generalized pigeonhole principle:

$$n = m \times (k-1) + 1 = 3 \times (1-1) + 1 = 4$$

Therefore, a minimum of 4 numbers must be selected to guarantee that at least one pair adds up to 7.

Example. How many cards must be selected from a standard deck of 52 cards to guarantee that at least 3 cards of the same suit are selected?
There are 4 suits in a standard deck of cards (hearts, diamonds, clubs, spades), so we have $m = 4$ suits. To ensure that at least one suit has at least 3 cards selected, we can use the generalized pigeonhole principle:

$$n = m \times (k-1) + 1 = 4 \times (3-1) + 1 = 9$$

Therefore, a minimum of 9 cards must be selected to guarantee that at least 3 cards of the same suit are selected.

Example. How many cards must be selected from a standard deck of 52 cards to guarantee that at least 3 hearts are chosen?
There are 13 hearts in a standard deck of cards, so to ensure that at least 3 hearts are selected, we can consider the worst-case scenario where we select all non-heart cards first. There are $52 - 13 = 39$ non-heart cards. To guarantee that at least 3 hearts are selected, we need to select:

$$n = 39 + 3 = 42$$

Therefore, a minimum of 42 cards must be selected to guarantee that at least 3 hearts are chosen.

8.5 Combinatorial Proofs

Definition 8.5.1 (Combinatorial Proofs).
A combinatorial proof is a method of proving mathematical identities by counting the same set of objects in two different ways. This approach often provides a more intuitive understanding of the identity being proved. There are two main steps in a combinatorial proof:

- Showing that there is a bijection between the set being counted by the two side of the identity.
- Prove that both sides of the identity count the same objects but in different ways.

8.5.1 Bijection Principle

Theorem 8.5.1 (Bijection Principle). If there exists a bijection between two finite sets A and B, then the cardinalities of the sets are equal, i.e., $|A| = |B|$.

Proof. A bijection is a one-to-one correspondence between the elements of two sets. This means that for every element in set A, there is a unique element in set B, and vice versa. Since each element in A can be paired with exactly one element in B, the number of elements in both sets must be the same. Therefore, if there exists a bijection between A and B, it follows that $|A| = |B|$. □

Example. Prove that $C(n, k) = C(n, n - k)$ using a combinatorial proof.
Consider a set S with n elements. The left side of the identity, $C(n, k)$, counts the number of ways to choose k elements from the set S. The right side of the identity, $C(n, n - k)$, counts the number of ways to choose $n - k$ elements from the same set S.
There is a bijection between the two sets of choices: for every subset of k elements chosen from S, there is a corresponding subset of $n - k$ elements that are not chosen. This means that choosing k elements is equivalent to choosing which $n - k$ elements to leave out. Therefore, both sides of the identity count the same number of subsets, leading to the conclusion that:

$$C(n, k) = C(n, n - k)$$

8.5.2 Double Counting Principle

Theorem 8.5.2 (Double Counting Principle). If a set S can be counted in two different ways, then the two counts must be equal.

Proof. Let S be a set that can be counted in two different ways, resulting in counts A and B. Since both counts represent the same set S, they must be equal. Therefore, we have:

$$A = B$$

□

Example. Let's prove the identity $\sum_{k=0}^{n} C(n, k) = 2^n$ using a combinatorial proof.
The left side of the identity, $\sum_{k=0}^{n} C(n, k)$, counts the total number of subsets of a set with n elements. This is because $C(n, k)$ represents the number of ways to choose k elements from the set, and summing over all possible values of k gives the total number of subsets.
The right side of the identity, 2^n, counts the same set of subsets by considering that each element in the set can either be included in a subset or not. Since there are n elements, and each element has 2 choices (to be included or not), the total number of subsets is 2^n.

Since both sides of the identity count the same set of subsets, we conclude that:

$$\sum_{k=0}^{n} C(n,k) = 2^n$$