CSE 191
Introduction to Discrete Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Course Summary and Review
Final Exam Logistics

● The final is Tuesday 5/16/23 in **Davis 101** from 3:30PM to 6:30PM
  ○ If you have any official conflicts (2 exams at same time, or 3 same day) please let me know ASAP
  ○ Seating will be randomized
  ○ All bags/electronics will be placed in the front of the room

● **What's provided:** Equivalence laws and inference rules

● **What to bring:**
  ○ UB ID card
  ○ Pen/pencil
  ○ One 8.5x11 cheat sheet (front and back)
Topic List

Emphasis on bolded topics (they were not covered by the midterm)

1. Propositional Logic
2. Logical Equivalence
3. Predicates and Quantifiers
4. Logical Reasoning/Proofs
5. Sets
6. Functions and Relations
7. Sequences
8. Counting
9. Graphs
10. Finite Automata
Propositional Logic
[Chapter 1.1, 1.2]
A proposition is a declarative statement

- Must be either **TRUE** (T) or **FALSE** (F)
  - Cannot be both...
  - Referred to as the **truth value** of the proposition

- An opinion of a specific person is a proposition
  - Their opinion would determine the true value

- The bits 0/1 are used for F/T
  - Digital logic uses 0/1, LOW/HIGH, or OFF/ON
  - Computers use bits and logic gates for **all** computation
Propositional Variables

**Propositional variables** are variables that represent propositions

- Commonly used letters are \( p, q, r, s, \ldots \)
  - Alternatively, the first letter of what we are trying to represent
- May be associated a specific proposition or left as a placeholder for an arbitrary proposition

**Compound propositions** are formed by using propositional variables and logical operators

- A compound proposition is itself a proposition
Logical Operators allow combining propositions into new ones
- Going forward: combine propositions to form new ones
- Going backward: decompose proposition into atomics

Example Compound Proposition

If I am at work, then I am wearing sneakers

Logical operator (if ..., then ...)
Let $p$ be a proposition.

The negation of $p$, denoted by $\neg p$ (or sometimes $\bar{p}$) is the statement:

"It is not the case that $p$"

- $\neg p$ is a new proposition, read as "not $p$"
- $\neg$ is referred to as the negation operator. It is a **unary** operator
  - Unary operators only operate on one proposition
- The truth value of $\neg p$ is the opposite of the truth value of $p$
Let $p$ and $q$ be propositions.

The *conjunction* of $p$ and $q$, denoted by $p \land q$ is the statement:

"$p$ and $q$"

and is only TRUE when $p$ and $q$ are both TRUE, and is FALSE otherwise.
Let $p$ and $q$ be propositions.

The *disjunction of $p$ and $q$*, denoted by $p \lor q$ is the statement:

"$p$ or $q$"

and is TRUE when $p$ is TRUE, $q$ is TRUE, or both are TRUE

$p \lor q$ is FALSE only when both $p$ and $q$ are FALSE
Let \( p \) and \( q \) be propositions.

The exclusive or of \( p \) and \( q \), denoted by \( p \oplus q \) (read XOR) is the statement:

"\( p \) or \( q \), but not both"

and is TRUE when exactly one of \( p \) and \( q \) is TRUE, and FALSE otherwise.
Let $p$ and $q$ be propositions.

The *implication* of $p$ on $q$, denoted by $p \rightarrow q$ is the statement:

"$p$ implies $q$" or "if $p$, then $q$" and is FALSE when $p$ is TRUE, $q$ is FALSE, and TRUE otherwise

$p$ is called the *hypothesis* or *antecedent* or *precedent*

$q$ is called the *conclusion* or *consequence*
Let $p$ and $q$ be propositions.

The bidirectional implication of $p$ on $q$, denoted by $p \iff q$ is the statement:

"$p$ if and only if $q$"

and is only TRUE when $p$ and $q$ have same truth value, FALSE otherwise.
A **Truth Table** lists all possible combinations of truth values of the operands, as well as the resulting truth value in the rightmost column.
Truth Tables: Negation Operator

- The negation operator has a single operand
  - This operand can either be TRUE or FALSE
- The truth value of $\neg p$ is the opposite of the truth value of $p$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

Truth table for negation
## Truth Tables: Binary Logical Operators

Conjunction/AND

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \land q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
## Truth Tables: Binary Logical Operators

### Disjunction/OR

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \lor q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

### Exclusive Or/XOR

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \oplus q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
**Truth Tables: Binary Logical Operators**

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \rightarrow q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

*Implication/If ..., then ...*

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \iff q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

*Bidirectional Implication/IFF*
Logical Equivalence
[Chapter 1.3]
A **tautology** is a compound proposition that is ALWAYS TRUE, no matter what the truth values of the propositional variables that occur in it.

A **contradiction** is a compound proposition that is ALWAYS FALSE.

A **contingency** is a compound proposition that is neither a contradiction or a tautology. There at least one assignment of truth values to the atomics that can result in TRUE, and at least one that can result in FALSE.
Logical Equivalence

Two propositions, \( p \) and \( q \) are **logically equivalent** if \( p \Leftrightarrow q \) is a tautology

- In other words, \( p \) and \( q \) are logically equivalent if their truth values in their truth table are all the same
- Two compound propositions are logically equivalent if their truth values agree for all combinations of the truth values of their atomics
- We write equivalence as \( p \equiv q \)
  - \( \equiv \) is NOT a logical operator
  - \( p \equiv q \) is NOT a compound proposition
## Logical Equivalence Rules

<table>
<thead>
<tr>
<th>Equivalence</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \land T \equiv p ) ( p \lor F \equiv p )</td>
<td>Identity laws</td>
</tr>
<tr>
<td>( p \lor T \equiv T ) ( p \land F \equiv F )</td>
<td>Domination laws</td>
</tr>
<tr>
<td>( p \lor p \equiv p ) ( p \land p \equiv p )</td>
<td>Idempotent laws</td>
</tr>
<tr>
<td>( \neg(\neg p) \equiv p )</td>
<td>Double negation law</td>
</tr>
<tr>
<td>( p \lor q \equiv q \lor p ) ( p \land q \equiv q \land p )</td>
<td>Commutative laws</td>
</tr>
<tr>
<td>( (p \lor q) \lor r \equiv p \lor (q \lor r) ) ( (p \land q) \land r \equiv p \land (q \land r) )</td>
<td>Associative laws</td>
</tr>
<tr>
<td>( p \lor (q \land r) \equiv (p \lor q) \land (p \lor r) ) ( p \land (q \lor r) \equiv (p \land q) \lor (p \land r) )</td>
<td>Distributive laws</td>
</tr>
<tr>
<td>( \neg(p \lor q) \equiv \neg p \land \neg q ) ( \neg(p \land q) \equiv \neg p \lor \neg q )</td>
<td>De Morgan's laws</td>
</tr>
<tr>
<td>( p \lor (p \land q) \equiv p ) ( p \land (p \lor q) \equiv p )</td>
<td>Absorption laws</td>
</tr>
<tr>
<td>( p \lor \neg p \equiv T ) ( p \land \neg p \equiv F )</td>
<td>Negation laws</td>
</tr>
</tbody>
</table>
**Logical Equivalence Rules**

### Equivalences with Implication

<table>
<thead>
<tr>
<th>Equivalences with Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \to q \equiv \neg p \lor q )</td>
</tr>
<tr>
<td>( p \to q \equiv \neg q \to \neg p )</td>
</tr>
<tr>
<td>( p \lor q \equiv \neg p \to q )</td>
</tr>
<tr>
<td>( p \land q \equiv \neg (p \to \neg q) )</td>
</tr>
<tr>
<td>( \neg (p \to q) \equiv p \land \neg q )</td>
</tr>
<tr>
<td>( (p \to q) \land (p \to r) \equiv p \to (q \land r) )</td>
</tr>
<tr>
<td>( (p \to q) \land (q \to r) \equiv (p \lor q) \to r )</td>
</tr>
<tr>
<td>( (p \to q) \lor (p \to r) \equiv p \to (q \lor r) )</td>
</tr>
<tr>
<td>( (p \to q) \lor (q \to r) \equiv (p \land q) \to r )</td>
</tr>
</tbody>
</table>

### Equivalences with Bidirectional Implication

<table>
<thead>
<tr>
<th>Equivalences with Bidirectional Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \leftrightarrow q \equiv (p \to q) \land (q \to p) )</td>
</tr>
<tr>
<td>( p \leftrightarrow q \equiv q \leftrightarrow p )</td>
</tr>
<tr>
<td>( p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q )</td>
</tr>
<tr>
<td>( p \leftrightarrow q \equiv (p \land q) \lor (\neg p \land \neg q) )</td>
</tr>
<tr>
<td>( \neg (p \leftrightarrow q) \equiv p \leftrightarrow \neg q )</td>
</tr>
</tbody>
</table>
Proving Logical Equivalence

- By using equivalence laws, we can prove two propositions are logically equivalent without having to construct large truth tables.
- The logical equivalences shown in the tables can be used to construct additional logical equivalences.
Proving Logical Equivalence

In General:

- Each line should be equivalent to the previous
- Each line should list the law that led to it
  - Exactly one law applied per line
- Start with LHS and go until you reach the RHS

**Note:** Logical equivalence proofs are very exact. Later proofs will be less restrictive.
A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true.

A compound proposition is **unsatisfiable** when no such assignment exists.

- A compound proposition is unsatisfiable iff its negation is a tautology.
- An assignment of truth values that make a compound proposition true is called a solution to that satisfiability problem.
Predicates and Quantifiers
[Chapter 1.4, 1.5]
A **predicate** is a *function* that takes some *variable(s) as arguments*; it returns either TRUE or FALSE, but never both, depending on the combination of the values passed as arguments.

**Example:** $P(x)$: $x$ is an even number.

- $P$ is the function, $x$ is the variable
- $P(x)$ is the value of the predicate $P$ at $x$
Given a predicate, $P(x)$, the **domain of discourse** (often just called the **domain**) is the set of all possible values for the variable $x$.

- Predicates with multiple variables may have:
  - Multiple domains of discourse (one for each variable)
  - A single domain of discourse for all variables
Quantifiers

**Quantification** expresses the extent to which a predicate is true over a range of elements. For example, in English: all, some, none, many, few, ...
Suppose $P(x)$ is a predicate on some domain, $D$.

The **universal quantification** of $P(x)$ is the *proposition*:

"$P(x)$ is true for all $x$ in the domain of discourse $D$." 

Written as: $\forall x, P(x)$
Read as: "For all $x$, $P(x)$" or "For every $x$, $P(x)$"

$\forall x, P(x)$ is TRUE if $P(x)$ is TRUE for **every** $x$ in $D$.

$\forall x, P(x)$ is FALSE if $P(x)$ is FALSE for **some** $x$ in $D$. 
Existential Quantification

Suppose $P(x)$ is a predicate on some domain, $D$.

The **existential quantification** of $P(x)$ is the proposition:

"$P(x)$ is true for some $x$ in the domain of discourse $D"."

Written as: $\exists x, P(x)$

Read as: "There exists an $x$ such that, $P(x)$" or "For some $x$, $P(x)$"

$\exists x, P(x)$ is TRUE if $P(x)$ is TRUE for **some** $x$ in $D$.

$\exists x, P(x)$ is FALSE if $P(x)$ is FALSE for **every** $x$ in $D$.
The occurrence of a variable, $x$, is said to be **bound** when a quantifier is used on that variable.

The occurrence of a variable, $x$, is said to be **free** when it is not bound by a quantifier or set to a particular variable.

The part of a logical expression to which a quantifier is applied is called the **scope** of this quantifier.

*Note: A variable is free if it is outside of the scope of all quantifiers in the formula that specify this variable.*
Quantifier Negation Rule

**Quantifier Negation**

In general we have, for any predicate \( P(x) \):

\[
\neg \forall x, P(x) \equiv \exists x, \neg P(x) \quad \text{and} \quad \neg \exists x, P(x) \equiv \forall x, \neg P(x)
\]

**De Morgan's Law for Quantifiers**

<table>
<thead>
<tr>
<th>Negation</th>
<th>Equivalent statement</th>
<th>Negation is TRUE when...</th>
<th>Negation is FALSE when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg \exists x, P(x) )</td>
<td>( \forall x, \neg P(x) )</td>
<td>For every x, ( P(x) ) is FALSE</td>
<td>There is an x where ( P(x) ) is TRUE</td>
</tr>
<tr>
<td>( \neg \forall x, P(x) )</td>
<td>( \exists x, \neg P(x) )</td>
<td>There is an x where ( P(x) ) is FALSE</td>
<td>( P(x) ) is true for every x</td>
</tr>
</tbody>
</table>
A logical expression with more than one quantifier that bind different variables in the same predicate is said to have nested quantifiers. In order to evaluate them we must consider their ordering and scope.
The portion of the formula a quantifier covers is called the **scope**
- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

\[ \forall i \ \exists j, \ (P(i,j) \rightarrow \ \forall k, \ Q(k,j)) \]
Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**
- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

\[
\forall i \ \exists j, \ (P(i,j) \rightarrow \ \forall k, \ Q(k,j))
\]

The scope of \( \forall i \) is the entire formula
Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**
- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

\[ \forall i \exists j, (P(i,j) \rightarrow \forall k, Q(k,j)) \]

The scope of \( \exists j \) is the entire formula (other than \( \forall i \))
Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**
- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

\[ \forall i \exists j, (P(i,j) \rightarrow \forall k, Q(k,j)) \]

The scope of \( \forall k \) is the limited to \( Q(k,j) \)
The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

\[ \forall i \exists j, (P(i,j) \rightarrow \forall k, Q(k,j)) \]
Logical Reasoning/Proofs
[Chapter 1.6-1.8]
Logical Reasoning: What is it?

Suppose the following are TRUE statements:
1. You will buy your friend lunch if they drive you to work
2. They drove you to work

What can you conclude?
You will buy your friend lunch.

Note: This differs from logical equivalence
- Statements derived are not always equivalent
- Can derive new knowledge from multiple facts
Logical Reasoning: Arguments

**Arguments** are:
- a list of propositions, called **hypotheses** (also called **premises**)
- a final proposition, called the **conclusion**

\[ p_1 \land p_2 \land \ldots \land p_n \rightarrow c \]

An argument is **valid** if \((p_1 \land p_2 \land \ldots \land p_n) \rightarrow c\) is a tautology
- Otherwise, it is **invalid**
- **Fallacies** are incorrect reasonings which lead to invalid arguments
A **logical proof** of an argument is a sequence of steps, each of which consists of a proposition and a justification.

Each line should contain one of the following:
- a hypothesis (assumption)
- a proposition that is equivalent to a previous statement
- a proposition that is derived by applying an argument to previous statements

Justifications should state one of the following:
- hypothesis
- the equivalence law used (and the line it was applied to)
- the argument used (and the line(s) it was applied to)

The last line should be the conclusion.
Logical Reasoning: Invalid Argument

**Remember:** An argument is valid if \((p_1 \land p_2 \land \ldots \land p_n) \to c\) is a tautology.

Therefore to show it is invalid, we need a **counterexample**. A counterexample is a situation where the hypotheses are all TRUE, and the conclusion is FALSE.

**Example** consider the converse as an argument.

\[
\begin{align*}
  p \to q & \quad \text{Suppose } p: \text{FALSE and } q: \text{TRUE.} \\
  \therefore q \to p & \quad \text{Then } p \to q \text{ is TRUE, but } q \to p \text{ is FALSE.}
\end{align*}
\]

Therefore the argument is invalid.
# Logical Reasoning: Rules of Inference

<table>
<thead>
<tr>
<th>Rule of Inference</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{align*} \ p \ p \rightarrow q \ \therefore q \end{align*} ]</td>
<td>Modus Ponens</td>
</tr>
<tr>
<td>[ \begin{align*} \neg q \ p \rightarrow q \ \therefore \neg p \end{align*} ]</td>
<td>Modus Tollens</td>
</tr>
<tr>
<td>[ \begin{align*} p \rightarrow q \ q \rightarrow r \ \therefore p \rightarrow r \end{align*} ]</td>
<td>Hypothetical Syllogism</td>
</tr>
</tbody>
</table>
# Logical Reasoning: Rules of Inference

<table>
<thead>
<tr>
<th>Rule of Inference</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \lor q )</td>
<td>Disjunctive Syllogism</td>
</tr>
<tr>
<td>( \neg p )</td>
<td></td>
</tr>
<tr>
<td>( \therefore q )</td>
<td></td>
</tr>
<tr>
<td>( p )</td>
<td>Addition</td>
</tr>
<tr>
<td>( \therefore p \lor q )</td>
<td></td>
</tr>
<tr>
<td>( p \land q )</td>
<td>Simplification</td>
</tr>
<tr>
<td>( \therefore p )</td>
<td></td>
</tr>
</tbody>
</table>
## Logical Reasoning: Rules of Inference

<table>
<thead>
<tr>
<th>Rule of Inference</th>
<th>Name</th>
</tr>
</thead>
</table>
| \[
\begin{align*}
\quad p \\
\quad q
\end{align*}
\]
\[\therefore p \land q\] | Conjunction |
| \[
\begin{align*}
\quad p \lor q \\
\quad \neg p \lor r
\end{align*}
\]
\[\therefore q \lor r\] | Resolution  |
Mathematical Proofs

- A **mathematical proof** is usually "informal"
- More formal than everyday language, less formal than logical proofs
  - More than one rule may be used in a step
  - *(Some)* steps may be skipped
  - Axioms may be assumed
  - Rules for inference need not be explicitly stated
- Proofs must be a self-contained line of reasoning containing only:
  - facts (axioms)
  - Theorems, lemmas, corollaries (previously proven statements), or
  - statements derived from the above

You cannot use something as fact within a proof if you are not certain that it is
Some Terminology

- **Theorem**: statement that can be shown true
  - **Proposition**: less important theorem
  - **Lemma**: less important theorem used to prove other theorems
  - **Corollary**: theorem that trivially follows another theorem
- **Conjecture**: statement proposed to be true, but not yet proven
- **Axiom**: statement assumed to be true (does not need a proof)
- Most axioms, theorems, etc are universal over some domain
  - ie all perfect squares are non-negative
  - the domain should be clear from context, or explicitly stated
A proof by exhaustion for $p \rightarrow q$ starts by considering each element of the domain of discourse and showing the predicate is true.

Only useful when dealing with a small domain:
- Small is relative, but must be finite.
- Example: $\{2,4,6\}$ is a small and finite domain.

This is a special type of proof by cases.
Direct Proofs

A **direct proof** for $P(x) \rightarrow Q(x)$ starts by assuming $P(x)$ as fact, and finishes by establishing $Q(x)$.

It makes use of axioms, previously proven theorems, inference rules, etc.

Same approach as proving a logical argument is valid:
- $P(x)$ is the hypothesis
- $Q(x)$ is the conclusion
A proof by contraposition for $P(x) \rightarrow Q(x)$ is a proof where you:

1. Write a direct proof for $\neg Q(x) \rightarrow \neg P(x)$
2. Conclude that the contrapositive, $P(x) \rightarrow Q(x)$, is also true

**Proof Layout:**

Assume $\neg Q(x)$

Perform your derivations (using theorems, axioms, etc)

$\therefore \neg P(x)$

Since $\neg Q(x) \rightarrow \neg P(x)$ is TRUE, we may conclude that our original statement $P(x) \rightarrow Q(x)$ is also TRUE
Note that $p$ is logically equivalent to $\neg p \rightarrow (r \land \neg r)$

A **proof by contradiction** for $p$ is actually a proof for $\neg p \rightarrow (r \land \neg r)$ where you:

1. Write a proof starting with the assumption $\neg p$
2. Find some proposition $r$ where you can derive both $r$ and $\neg r$ to be TRUE (a contradiction)

**Proof Layout:**
Assume $\neg p$
Find something that breaks
$\therefore$ contradiction, so $p$ has to be true
Additional Review
[Chapter 1 Review/Summary Exercises]
Sets

[Chapter 2.1, 2.2]
A set is a collection of objects that do NOT have an order.

Each object is called an element or member of the set.

Notation:
- $e \in S$ means that $e$ is an element of $S$.
- $e \notin S$ means that $e$ is not an element of $S$. 
Common Sets

- **N** = \{1, 2, 3, \ldots\}: the set of natural numbers
  - Sometimes 0 is considered a member, which some disagree with
- **Z** = \{0, -1, 1, -2, 2, \ldots\}: the set of integers
- **Z\(^+\)** = \{1, 2, 3, \ldots\}: the set of positive integers
- **Q** = \{p/\(q\) | \(p \in Z\), \(q \in Z\), \(q \neq 0\}\}: the set of rational numbers
  - Numbers that can be written as a fraction of integers
- **Q\(^+\)** = \{x | x \(\in\) Q, x > 0\}: the set of positive rational numbers
- **R**: the set of real numbers
- **R\(^+\)** = \{x | x \(\in\) R, x > 0\}: the set of positive real numbers
- **C**: the set of complex numbers
Cardinality (for Finite Sets)

If a set $A$ contains exactly $n$ elements, where $n$ is a non-negative integer, then $A$ is a finite set.

$n$ is called the cardinality of $A$, denoted by $|A|$.

The empty set or null set is the set that contains no elements, denoted by $\emptyset$ or $\{\}$. It has size 0.
A set $A$ is a **subset** of $B$ if and only if every element of $A$ is also in $B$.

Denoted by $A \subseteq B$

If $A \subseteq B$, then $\forall x \in A, x \in B$

**Note:** for any set $A$, $\emptyset \subseteq A$ and $A \subseteq A$

If $A \subseteq B$ but $A \neq B$, then $A$ is a **proper subset** of $B$.

Denoted by $A \subset B$ or $A \varsubsetneq B$
Set Equality

**Fact:** Suppose $A$ and $B$ are sets. Then $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$

**To Prove Set Equality**

**Prove $A \subseteq B$:**
Assume $x$ in $A$

\[ \vdots \]
\[ \therefore x \text{ in } B \text{ as well} \]
Conclude that $A \subseteq B$

**Prove $B \subseteq A$:**
Assume $y$ in $B$

\[ \vdots \]
\[ \therefore y \text{ in } A \text{ as well} \]
Conclude that $B \subseteq A$

Conclude that since $A \subseteq B$ and $B \subseteq A$ then $A = B$
Set Union

The **union** of two sets, $A$ and $B$, is the set that contains exactly all elements that are in $A$ or $B$ (or in both)

- Denoted by $A \cup B$
- Formally, $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$
The **intersection** of two sets, $A$ and $B$, is the set that contains exactly all elements that are in $A$ and $B$

- Denoted by $A \cap B$
- Formally, $A \cap B = \{ x | x \in A \text{ and } x \in B \}$
Set Intersection

The **intersection** of two sets, \( A \) and \( B \), is the set that contains exactly all elements that are in \( A \) and \( B \)

- Denoted by \( A \cap B \)
- Formally, \( A \cap B = \{ x \mid x \in A \text{ and } x \in B \} \)

Two sets are **disjoint** if their intersection is the empty set

**Principle of Inclusion-Exclusion**

\[ |A \cup B| = |A| + |B| - |A \cap B| \]
The **complement** of set $A$ is the set that contains exactly all the elements that are not in $A$.

- Denoted by $\bar{A}$
- Formally, $\bar{A} = \{ x \mid x \notin A \}$

\[ A \cup B \text{ is shaded} \]
Set Difference

The **difference** of set $A$ and set $B$ is the set that contains exactly all elements that are in $A$ but not in $B$

- Denoted by $A - B$ (or $A \setminus B$)
- Formally, $A - B = \{ x \mid x \in A \text{ and } x \notin B \} = A \cap \overline{B}$

$A - B$ is shaded → $A - (B \cup C)$ is shaded
Symmetric Difference

The **symmetric difference** of set $A$ and set $B$ is the set that contains all elements that are in exactly one of $A$ or $B$

- Denoted by $A \oplus B$ (or $A \bigtriangleup B$)
- Formally, $A \oplus B = (A - B) \cup (B - A)$

$A \oplus B$ is shaded

$A \oplus B \oplus C$ is shaded

It includes values that are in an odd number of sets, ie $\{ x | x \in A \oplus x \in B \oplus x \in C \}$
The **power set** of set $A$ is the set of all possible subsets of $A$

Denoted by $\mathcal{P}(A)$

In general, $|\mathcal{P}(A)| = 2^{|A|}$

For any set $A$, it is always the case that:

- $\emptyset \in \mathcal{P}(A)$ (the empty set is a subset of $A$ ...and every other set)
- $A \in \mathcal{P}(A)$ ($A$ is a subset of itself...every elements of $A$ is in $A$)
Imposing Order on Elements

Sometimes order is important...

How can we impose order on elements?

An ordered n tuple \((a_1, a_2, \ldots, a_n)\) has \(a_1\) as its first element, \(a_2\) as its second, \(\ldots,\) and \(a_n\) as its \(n^{th}\) element.

Order is important for tuples. Assume \(a_1 \neq a_2\)

- \((a_1, a_2) \neq (a_2, a_1)\) ← tuple comparison
- \(\{a_1, a_2\} = \{a_2, a_1\}\) ← set comparison
The **Cartesian product** of two sets $A_1$ and $A_2$ is defined as the set of ordered tuples $(a_1, a_2)$ where $a_1 \in A_1$ and $a_2 \in A_2$

- Denoted by $A_1 \times A_2$
- Formally, $A_1 \times A_2 = \{(a_1, a_2) \mid a_1 \in A_1 \text{ and } a_2 \in A_2\}$
- We say "$A_1 \text{ cross } A_2$"

René Descartes
Strings

An **alphabet** is a *non-empty finite* set of symbols.

A **string** is a finite sequence of symbols from an alphabet.

- Shorthand for a tuple from the Cartesian power of an alphabet.

The number of characters in a string is called the **length** of the string.

- The length of string $s$ is denoted by $|s|$. 
Two sets $A$ and $B$ are **disjoint** iff $A \cap B = \emptyset$.

A sequence of sets, $A_1, A_2, A_3, \ldots, A_n$ are **pairwise disjoint** if:
for any $i, j \in \{1,2,3,\ldots,n\}$, where $i \neq j$, we have $A_i \cap A_j = \emptyset$.

Symbolically we write $\forall i,j \in \{1,2,3,\ldots,n\}: [(i \neq j) \rightarrow (A_i \cap A_j = \emptyset)]$.
A **partition** of a non-empty set $A$ is a list of one or more non-empty subsets of $A$ such that each element of $A$ appears in exactly one of the subsets.

Formally, a partition of $A$ is a list of sets, $A_1, A_2, ..., A_k$ such that:
1. $\forall i \in [1, k]: A_i \neq \emptyset$ (the sets are non-empty)
2. $\forall i \in [1, k]: A_i \subseteq A$ (the sets are subsets of $A$)
3. $\forall i, j \in [1, k]: i \neq j \rightarrow A_i \cap A_j = \emptyset$ (the sets are pairwise disjoint)
4. $A = A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_k$
Functions and Relations
[Chapter 2.3, 9.1, 9.5, 9.6]
A **binary relation** between two sets $A$ and $B$ is any set $R \subseteq A \times B$

A binary relation **from $A$ to $B$** is a set $R$ of ordered pairs, where the first element of each ordered pair comes from $A$ and the second from $B$

- For any $a \in A$ and $b \in B$ we say that $a$ is related to $b$ iff $(a,b) \in R$
- Denoted by $a R b$

**Note:** a relation is a binary predicate $R(a,b)$: "$a$ is related to $b"
Example

Consider the set of students, $S = \{ \text{Alice, Bob, Carol, Don} \}$, and the set of courses, $C = \{ \text{CSE115, CSE116, CSE191} \}$.

- Alice, Bob, and Carol are enrolled in CSE115.
- Don is enrolled in CSE116.
- Alice and Don are enrolled in CSE191.

This is called an arrow diagram. It is a visual representation of a binary relation.
Example

Consider the set of students, \( S = \{ \text{Alice, Bob, Carol, Don} \} \), and the set of courses, \( C = \{ \text{CSE115, CSE116, CSE191} \} \).

We can also use **matrix representation** to describe \( E \):

\[
\begin{pmatrix}
\text{CSE115} & \text{CSE116} & \text{CSE191} \\
\text{Alice} & 1 & 0 & 1 \\
\text{Bob} & 1 & 0 & 0 \\
\text{Carol} & 1 & 0 & 0 \\
\text{Don} & 0 & 1 & 1
\end{pmatrix}
\]
The binary relation $R$ on a set $A$ is a subset of $A \times A$.

The set $A$ is called the **domain** of the binary relation.
A relation $R$ on set $A$ is called **reflexive** if every $a \in A$ is related to itself. Formally, $a R a$ for all $a \in A$.

**Example:** Consider the $\leq$ relation on $\mathbb{Z}$.
A relation $R$ on set $A$ is called **symmetric** if for every $a R b$, we also have that $b R a$.

**Example:** Consider the $=$ relation on $\mathbb{Z}$.

A relation $R$ on set $A$ is called **anti-symmetric** if for all $a, b \in A$: $a R b$ and $b R a$ implies that $a = b$.

**Example:** Consider the $\leq$ relation on $\mathbb{Z}$.
A relation $R$ on set $A$ is called **transitive** if for all $a,b,c \in A$:

$a \mathrel{R} b$ and $b \mathrel{R} c$ implies $a \mathrel{R} c$.

**Example:** Consider the $<$ relation on $\mathbb{Z}$
A relation $R$ on a set $A$ is called a **partial order** if it is reflexive, transitive, and antisymmetric.

$a R b$ is denoted $a \leq b$ for partial a ordering $R$

- We read $a \leq b$ as "$a$ is at most $b$" or "$a$ precedes $b$"
- A domain, $A$, with a partial ordering $\leq$ can be treated as the object $(A, \leq)$
  - $(A, \leq)$ is called a **partially ordered set** or **poset**
Comparable Elements and Total Ordering

Elements $x$ and $y$ are **comparable** if $x \leq y$ or $y \leq x$ (or both)

A partial order is a **total order** if every pair of elements in the domain are comparable.

In our previous example, $(\mathbb{Z}, R)$ is a total order
- It is a partial order, and for every $x, y \in \mathbb{Z}$, $x R y$ or $y R x$
- We say that $R$ is a total ordering of $\mathbb{Z}$
A relation $R$ on a set $A$ is called an **equivalence relation** if it is reflexive, transitive, and symmetric.

$a R b$ is denoted $a \sim b$ for an equivalence relation $R$

- We read $a \sim b$ as "$a$ is equivalent to $b$"
We can partition the domain of an equivalence relation into equivalent elements. These partitions are called **equivalence classes**.

If $e \in D$ then the equivalence class containing $e$ is denoted $[e]$

$[e] = \{x \mid x \in D, x \sim e\}$
Let $A$ and $B$ be nonempty sets. A function $f$, from $A$ to $B$ is an assignment of exactly one element of $B$ to each element of $A$.

Denoted by $f: A \rightarrow B$

We write $f(a) = b$ if $b$ is the unique element of $B$ assigned by $f$ to the element $a$ of $A$.

The set $A$ is the domain of $f$.

The set $B$ is the codomain of $f$. 
Function Range

If $f$ is a function from $A$ to $B$, the set $\text{range}(f) = \{ y \mid \exists x \in A, f(x) = y \}$ is called the range of $f$

It is the set of all values in the codomain that have an element from the domain mapped to it

- For any function $f: A \rightarrow B$, $\text{range}(f) \subseteq B$
- It does not have to be the whole codomain
Injective Functions

A function \( f: A \rightarrow B \) is **injective** if \( \forall x_1, x_2 \in A, (f(x_1) = f(x_2) \rightarrow x_1 = x_2) \)

Also known as **one-to-one** or **1-1**

- Each element in the domain is mapped to a unique element from the codomain (no element in the codomain is hit twice)
- To prove a function is 1-1
  - Take an arbitrary \( x \) and \( y \) such that \( f(x) = f(y) \)
  - Conclude that \( x = y \)
- To prove a function is not 1-1
  - Find a counterexample where \( x \neq y \) but \( f(x) = f(y) \)
Surjective Functions

A function $f: A \rightarrow B$ is **surjective** if $\forall y \in B, \exists x \in A, f(x) = y$

Also known as **onto**

- Every element in the codomain has an element that maps to it
- To prove a function is onto:
  - Take arbitrary $y$ in the codomain
  - Find the value of $x$ in the domain such that $f(x) = y$
- To prove a function is not onto:
  - Find a counterexample, element $y$ in codomain s.t. no element maps to it
A function $f: A \rightarrow B$ is **bijective** if it is injective and surjective.

A bijective function is called a **bijection**, or a **one-to-one correspondence**.
For any function $f: A \rightarrow B$, the inverse mapping of $f$, denoted by $f^{-1}$, is defined by the mapping $f^{-1}: B \rightarrow A$ where: $f^{-1} = \{ (y, x) | (x, y) \in f \}$

If $f$ is a bijection then $f^{-1}$ is a function (otherwise it is just a mapping)
- $f^{-1}$ maps codomain elements of $f$ to domain elements of $f$
- If $f(x) = y$ then $f^{-1}(y) = x$
The floor function is the function $\text{floor} : \mathbb{R} \rightarrow \mathbb{Z}$ defined by

$$\text{floor}(x) = \max\{ y \mid y \in \mathbb{Z}, y \leq x \}$$

Evaluates to the maximum integer below the given number.

Denoted by: $\text{floor}(x) = \lfloor x \rfloor$

Examples

$\lfloor 4.5 \rfloor = 4$  \hspace{1cm}  $\lfloor 17 \rfloor = 17$

$\lfloor -8.7 \rfloor = -9$  \hspace{1cm}  $\lfloor \pi \rfloor = \lfloor 3.14159 \rfloor = 3$
Ceiling Function

The ceiling function is the function $\text{floor} : \mathbb{R} \rightarrow \mathbb{Z}$ defined by

$$\text{ceiling}(x) = \min\{ y \mid y \in \mathbb{Z}, y \geq x \}$$

Evaluates to the minimum integer above the given number.

Denoted by: $\text{ceiling}(x) = \lceil x \rceil$

**Examples**

$$\lceil 4.5 \rceil = 5$$  $$\lceil 17 \rceil = 17$$

$$\lceil -8.7 \rceil = -8$$  $$\lceil \pi \rceil = \lceil 3.14159 \rceil = 4$$
Divides

Let $x$ and $y$ be integers. Then $x$ divides $y$ if there is an integer $k$ s.t. $y = kx$.

Denoted by $x \mid y$

- $x$ does not divide $y$ is denoted by $x \nmid y$

If $x \mid y$, then we say:
- $y$ is a multiple of $x$
- $x$ is a factor or divisor of $y$
The Division Algorithm for $n \in \mathbb{Z}$ and $d \in \mathbb{Z}^+$ gives unique values $q \in \mathbb{Z}$ and $r \in \{0, \ldots, d - 1\}$.

- The number $q$ is called the quotient.
- The number $r$ is called the remainder.

The operations \texttt{div} and \texttt{mod} produce the quotient and the remainder, respectively, as a function of $n$ and $d$.

- $n \texttt{ div } d = q$
- $n \texttt{ mod } d = r$

In programming, $n \% d = r$ denotes $n \texttt{ mod } d = r$
Addition mod $n$

For any integer $n > 0$, $x \mod n$ can be seen as a function $\mod_n(x)$:
- $\mod_n : \mathbb{Z} \to \{0, 1, 2, \ldots, n-1\}$, where $\mod_n(x) = x \mod n$.

Addition mod $n$ is defined by adding two numbers and then applying $\mod_n$:
- All results in the range \{0, 1, \ldots, n - 1\}

Suppose $n = 7$

+ $\mod_7(4, 6) = (4 + 6) \mod 7 = 10 \mod 7 = 3$

+ $\mod_7(15, 17) = (15 + 17) \mod 7 = 32 \mod 7 = 4$

+ $\mod_7(8, 20) = (8 + 20) \mod 7 = 28 \mod 7 = 0$
Multiplication mod $n$

Multiplication \textit{mod} $n$ is defined by multiplying two numbers and then applying mod $n$.

- All results in the range \{0, 1, ..., $n - 1$\}

Suppose $n = 11$.

\[mod_{11}(4, 6) = (4 \times 6) \mod 11 = 24 \mod 11 = 2\]

\[mod_{11}(5, 7) = (5 \times 7) \mod 11 = 35 \mod 11 = 2\]

\[mod_{11}(8, 23) = (8 \times 23) \mod 11 = 184 \mod 11 = 8\]
Congruence Modulo

If \( a \) and \( b \) are integers and \( m \) is a positive integer, then \( a \) is congruent to \( b \) modulo \( m \) if \( m \) divides \( a - b \).

The notation \( a \equiv b \ (mod \ m) \) indicates that \( a \) is congruent to \( b \) modulo \( m \).
- \( a \equiv b \ (mod \ m) \) is a congruence.
- Indicates that \( a \) and \( b \) are in the same equivalence class.

Is 17 congruent to 5 modulo 6? Yes, because 6 divides 17 - 5.
- 17 \( mod \) 6 = 5; it is in the equivalence class for 5 in \( mod \) 6.
- 5 \( mod \) 6 = 5; it is in the equivalence class for 5 in \( mod \) 6.
Composition of Functions

If $f$ and $g$ are two functions, where $f: X \to Y$ and $g: Y \to Z$, the composition of $g$ with $f$, denoted by $g \circ f$, is the function:

$$(g \circ f): X \to Z, \text{ s.t. for all } x \in X, (g \circ f)(x) = g(f(x))$$
Sequences
[Chapter 2.4, 5.1, 5.2]
Sequences: Terminology

A **sequence** is created by a special type of function with a domain of consecutive integers...ie no gaps in the domain

**OK:** \( \mathbb{N}, \mathbb{Z}^+, \mathbb{Z}^+ \cup \{0\}, \mathbb{Z}, \{1,2,34,5\} \)

**Not OK:** \{1,3,5,7\}, \{ x \in \mathbb{N} | x \text{ is even} \}
Increasing Sequences

A sequence \( \{a_k\} \) is **increasing** if, \( \forall i, a_i < a_{i+1} \)

A sequence \( \{a_k\} \) is **non-decreasing** if, \( \forall i, a_i \leq a_{i+1} \)

d\(_k\) = \( k \) for \( 1 \leq k \leq 10 \) is increasing

e\(_k\) = 2\(_k\) for \( k \geq 1 \) is increasing

f\(_k\) = 2\(^k\) for \( k \geq 0 \) is increasing

What about the sequence \( \{h_k\} = 1, 2, 2, 2, 3 \)？ **Non-Decreasing**
A sequence \( \{a_k\} \) is **decreasing** if, \( \forall i, a_i > a_{i+1} \)

A sequence \( \{a_k\} \) is **non-increasing** if, \( \forall i, a_i \geq a_{i+1} \)

\( s_k = 10 - k \) for \( 1 \leq k \leq 10 \) is decreasing (and non-increasing)

\( t_k = -2k \) for \( k \geq 1 \) is decreasing (and non-increasing)

\( u_k = 2^{-k} \) for \( k \geq 0 \) is decreasing (and non-increasing)

What about the sequence \( \{vh_k\} = 3, 2, 2, 2, 1? \) **Non-Increasing**
A **geometric sequence** is a sequence formed by successively multiplying the initial term by a fixed number called the **common ratio**.

**Examples:**

\{a_k\} is 1, -1, 1, -1, 1, -1, 1, -1, ...

\[
\rightarrow a_k = a_0 \cdot r^k = 1 \cdot (-1)^k \text{ for all } k \geq 0
\]

\{b_k\} is 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...

\[
\rightarrow b_k = b_0 \cdot r^k = 1 \cdot \left(\frac{1}{2}\right)^k \text{ for all } k \geq 0
\]

What are the explicit formulas for the above sequences?

For any geometric sequence \{s_k\} with initial term \(s_0\) and common ratio \(r\):

\[
s_k = s_0 \cdot r^k, \text{ for } k \geq 0
\]
An **arithmetic sequence** is a sequence formed by successively adding a fixed number, called the **common difference**, to the initial term.

**Examples:**

\( \{a_k\} \) is 5, 15, 25, 35, 45, ... \( \rightarrow a_k = a_0 \cdot kd = 5 + 10 \cdot k \) for all \( k \geq 0 \)

\( \{b_k\} \) is 49, 42, 35, 28, 21, ... \( \rightarrow b_k = b_0 \cdot kd = 49 + (-7)k \) for all \( k \geq 0 \)

What are the explicit formulas for the above sequences?

For any arithmetic sequence \( \{s_k\} \) with initial term \( s_0 \) and common diff \( d \):

\[ s_k = s_0 + kd, \text{ for } k \geq 0 \]
Summations

**Summation notation** is used to express the sum of terms in a numerical sequence.

Consider the sequence: $a_0, a_1, a_2, a_3, \ldots, a_k$

We can express the sum of all elements in the sequence as:

$$\sum_{i=0}^{k} a_i$$

What this represents is:

$$\sum_{i=0}^{k} a_i = a_0 + a_1 + a_2 + \ldots + a_k$$
A recurrence relation for the sequence \( \{a_n\} \) is an equation that expresses \( a_n \) in terms of one or more of the previous terms \( (a_0, a_1, a_2, \ldots, a_{n-1}) \), for all integers \( n \) with \( n \geq n_0 \), where \( n_0 \) is a nonnegative integer.

- A recurrence relation is said to recursively define a sequence
- It may have one or more initial terms
- A sequence is called a solution of a recurrence relation if its terms satisfy the recurrence relation
Mathematical Induction

**Principle of Mathematical Induction:**

Let $P(n)$ be a statement defined for any $n \in \mathbb{N}$. If the following hold:

- $P(1)$ is true
- For all $k \in \mathbb{N}$, $P(k) \rightarrow P(k + 1)$

Then $P(n)$ is true for all $n \in \mathbb{N}$

**Note:** We can relax this to apply to any domain $D$ of consecutive integers.

To prove the inductive step, assume $P(k)$, then derive $P(k + 1)$
Strong Induction

**Principle of Strong Mathematical Induction**

Let $a, b$ be integers with $a \leq b$

Let $P(n)$ be a statement defined for any integer $n \geq a$

Then $P(n)$ is true for all $n \geq a$ if the following two conditions hold:

1. $P(a), P(a + 1), \ldots, P(b)$ are all individually true (the base cases)
2. For all $k \geq b$, $P(a) \land P(a + 1) \land \ldots \land P(k) \rightarrow P(k + 1)$ (the inductive case)
Proof Template

To formally prove something via strong induction, you must do all of the following:

1. Express the statement being proved in the form \( \forall n \geq a, P(n) \) for a fixed integer \( a \)
2. Prove the Base Cases: show that \( P(a), P(a + 1), \ldots, P(b) \) are all true.
3. Prove the Inductive Case
   a. Inductive Hypothesis: Assume \( P(i) \) is true for all \( i \), where \( a \leq i \leq k \) for an arbitrary \( k \geq b \)
   b. State what must be proved under this assumption; write out \( P(k + 1) \)
   c. Prove the statement \( P(k + 1) \) is true by using the inductive assumption
   d. Clearly identify the conclusion
4. Now that you have proven the Base Case and Inductive Case, conclude that \( P(n) \) is true for all \( n \geq a \) by the principle of strong mathematical induction
Counting
[Chapter 6.1-6.3]
The Product Rule

Suppose that a procedure can be broken down into a sequence of 2 tasks.

If there are $n_1$ ways to do the first task, and for each of these ways of doing the first task, there are $n_2$ ways to do the second task, then there are $n_1 \cdot n_2$ ways to do the procedure.

The product rule can be phrased in terms of sets:

Let $A_1, A_2, ..., A_n$ be finite sets. Then $|A_1 \times A_2 \times ... \times A_n| = |A_1| \cdot |A_2| \cdot ... \cdot |A_n|$
The Sum Rule

If a task can be done either in one of \( n_1 \) ways OR in one of \( n_2 \) ways, where none of the of \( n_1 \) ways is the same as any of the \( n_2 \) ways, then there are \( n_1 + n_2 \) ways to do the task.

The sum rule can be phrased in terms of sets:

Let \( A_1, A_2, \ldots, A_n \) be mutually disjoint. Then \( |A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1| + |A_2| + \ldots + |A_n| \)
A permutation of a set of distinct objects is an ordered arrangement of these objects.

An ordered arrangement of $r$ elements of a set is called an $r$-permutation. The number of $r$-permutations of a set with $n$ elements is denoted by $P(n,r)$ or $nPr$. 
Let \( n \geq 0 \) be an integer. The **factorial** of \( n \), denoted by \( n! \) is defined by:

\[
n! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \ldots \cdot 2 \cdot 1
\]

**Note:** For convenience, we define \( 0! = 1 \).

We can also write it as a recurrence relation:

\[
a_0 = 1
\]

\[
a_n = n \cdot a_{n-1} \quad \text{for } n > 0
\]
If $n$ is a positive integer, and $r$ is an integer s.t. $1 \leq r \leq n$, then there are

$$P(n,r) = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \ldots \cdot (n - r + 1)$$

$r$-permutations of a set with $n$ distinct elements.
A **combination** of a set of *distinct* objects is an **unordered arrangement** of these objects. An **r-combination** is simply a subset with *r* elements.

The number of **r-combinations** of a set with *n* elements is denoted by $C(n,r)$ or $nCr$ or $\binom{n}{r}$. Sometimes referred to as *n* choose *r*. 
Combinations

Theorem

For any non-negative integers \( n \) and \( r \) s.t. \( 0 \leq r \leq n \):

\[
C(n, r) = \frac{n!}{r!(n - r)!} = \frac{n \cdot (n - 1) \cdots (n - r + 1)}{r \cdot (r - 1) \cdots 2 \cdot 1}
\]

...Therefore \( C(5, 3) = \frac{5!}{(3!2!)} = \frac{120}{6 \cdot 2} = 10 \)
Pigeonhole Principle: If you put $k$ pigeons into $n$ pigeonholes, with $k > n$, then at least one pigeonhole contains at least two pigeons.

In terms of functions: If $f: A \rightarrow B$ where the codomain has size $|B| = n$ and the domain $|A| = k$ where $k > n$, then $f$ must map at least 2 domain items to the same codomain element.
Pigeonhole Principle

**Generalized Pigeonhole Principle:**

If you put $k$ objects into $n$ boxes, then at least one box contains at least $\lceil k/n \rceil$ objects.

Basically, you cannot put a fraction of an item in a box (or more gruesomely...you cannot split up one pigeon into multiple boxes).

The fractional item gets rounded up (ceiling function)
Contrapositive Pigeonhole Principle

Contrapositive of the generalized pigeonhole principle:
Suppose you have $k$ elements and $n$ boxes. In order to guarantee that there is a box that contains at least $b$ items, $k$ must be at least $n*(b-1) + 1$. 
Graphs

[Chapter 10.1 - 10.5, 10.8, 11.1]
An (undirected) graph $G = (V, E)$ consists of $V$, a nonempty set of vertices (or nodes) and a set of edges.

- Each edge has one or two vertices associated with it, called endpoints.
- An edge, $\{u, v\}$ is said to connect its endpoints $u$ and $v$. 

**Vertices (V)**

1 3

**Edges (E)**

$\{1,3\}$

$\{1,2\}$

$\{2,4\}$

$\{3,4\}$
**Undirected vs Directed**

**Undirected Edge:** $\{u, v\}$ represented as a set
- Order doesn't matter ($\{u, v\} = \{v, u\}$)
- Represents a symmetric relationship

**Directed Edge:** $(u, v)$ represented as a tuple
- Order does matter, $(u, v)$ and $(v, u)$ are not the same
- Both edges may not exist
- Represents an asymmetric relationship (ie a one-way street)
A **Simple Graph** is a graph in which every edge connects two **different** vertices and where no two edges connect the same pair of vertices.
A graph that may have multiple edges connected the same vertices are called **multigraphs**.

- When there are \( m \) different edges associated with the same pair of vertices, \( \{u, v\} \), we say that edge \( \{u, v\} \) has **multiplicity** \( m \).
The edges that connect a given vertex to itself are called **loops** or sometimes **self-loops**.

- Graphs that may include loops and/or multiple edges between the same pair of vertices are sometimes called **pseudographs**.
A directed graph (or digraph) \((V, E)\) consists of a nonempty set of vertices \(V\) and a set of directed edges (or arcs) \(E\).

- Each directed edge is associated with an ordered pair of vertices.
- The directed edge \((u, v)\) is said to start at \(u\) and end at \(v\).
More Terminology

In an undirected graph:

Two vertices are **adjacent** (or **neighbors**) if they are endpoints of an edge.

An edge is **incident with** (or **connecting**) its endpoints.

The **degree of a vertex** is \( v \), denoted by \( \text{deg}(v) \) is the number of edges incident with \( v \).

**Note:** A **loop** is an edge of the form \( \{v,v\} \), and it adds to the degree of \( v \) twice.
The set of all neighbors of a vertex $v$ of $G = (V, E)$, denoted by $N(v)$, is called the **neighborhood** of $v$.

If $A$ is a subset of $V$, then the neighborhood of $A$, or $N(A)$ is the set of all vertices that are adjacent to at least one vertex in $A$. So $N(A) = \bigcup_{v \in A} N(v)$.
In a graph with directed edges

The **in-degree** of a vertex \( v \), denoted by \( \text{deg}^{-}(v) \), is the number of edges with \( v \) as their terminal (or ending) vertex.

The **out-degree** of a vertex \( v \), denoted by \( \text{deg}^{+}(v) \), is the number of edges with \( v \) as their initial (or starting) vertex.

**Note:** A loop at vertex \( v \) contributes 1 to both the in and out degree of \( v \).
A **complete graph on \( n \) vertices**, denoted by \( K_n \), is a simple graph that contains exactly one edge between each pair of distinct vertices.
A **subgraph** of a graph $G=(V,E)$ is a graph $H=(W,F)$ where $W \subseteq V$ and $F \subseteq E$.

A subgraph, $H$ of $G$ is a **proper subgraph** of $G$ if $H \not\cong G$.

A **spanning subgraph** is a subgraph in which $W = V$.

*Diagrams:*

- $K_5$
- A subgraph of $K_5$
- A spanning subgraph of $K_5$
In the **Adjacency List** representation of a graph, each vertex has a list of all of its neighbors.

**Note:** if the graph is undirected, then if \(a\) is in \(b\)'s list of neighbors, \(b\) must also be in \(a\)'s list of neighbors.
The **Adjacency Matrix** for a graph $M$ with $n$ vertices is an $n$ by $n$ matrix, whose entries are 0 or 1, indicating if an edge is present.

- $M_{i,j} = 1$ iff $\{i, j\}$ is an edge in the graph
- The vertices of $M$ are labeled with integers in the range 1 to $n$
- If $M$ is undirected, $M_{i,j} = M_{j,i}$ because edge $\{i, j\} = \text{edge} \ \{j, i\}$

![Diagram of a graph with labeled vertices and edges]!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Handshake Theorem

**Theorem**

For any undirected graph, $G = (V, E)$: 

$$\sum_{v \in V} \text{deg}(v) = 2|E|$$

**Note:** If $V = \{v_1, v_2, ..., v_n\}$ then

$$\sum_{v \in V} \text{deg}(v) = \text{deg}(v_1) + \text{deg}(v_2) + ... + \text{deg}(v_n)$$
The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a one-to-one and onto function $f$ from $V_1$ to $V_2$ with the property that $a$ and $b$ are adjacent in $G_1$ if and only if $f(a)$ and $f(b)$ are adjacent in $G_2$ for all $a$ and $b$ in $V_1$.

- Such a function is called an isomorphism.
- Two simple graphs that are not isomorphic are called nonisomorphic.
A **walk** from \( s \) to \( t \) is a sequence of vertices \( s, v_1, v_2, \ldots, v_k, t \) such that there is an edge between any two consecutive vertices in the list.

- If the first and last vertices are different it's an **open walk**.
- If they are the same, then it's a **closed walk**.

A **trail** from \( s \) to \( t \) is an open walk that has no repeated edges.

A **path** from \( s \) to \( t \) is a trail from \( s \) to \( t \) where all vertices are unique.

- Note that a path is a special case of a trail.
A circuit is a closed walk that has no repeated edges.

A cycle is a circuit with length at least three such that there are no repeated vertices other than the first and the last.

- A closed path is a cycle.
Euler Trail and Circuit

An **Euler circuit** in a graph $G$ is a simple circuit containing every edge of $G$.

An **Euler trail** in a graph $G$ is a trail that visits every edge of $G$ exactly once.

**Theorem**

A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has an even degree.
A path in a graph $G$ that passes through every vertex exactly once is called a Hamiltonian path.

A cycle in a graph $G$ that passes through every vertex exactly once is called a Hamiltonian cycle.
Connectivity

A node $s$ is **connected** to $t$ if there is a path from $s$ to $t$

A node $s$ is **isolated** if there is no other vertex connected to $s$
A set of vertices in a graph is **connected** if every pair of vertices in the set is connected.

A graph is said to be **connected** if the entire set of vertices is connected.

A graph that is not connected is **disconnected**.

A **connected component** is a **maximal** set of connected vertices.

**Note:** A disconnected graph can be split into more than one connected component.
Graph Coloring

Let $G = (V, E)$ be an undirected graph and $C$ be a finite set of colors. A valid coloring of $G$ is a function $c: V \rightarrow C$ such that for $\forall \{x, y\} \in E$, $c(x) \neq c(y)$
- If $|C| = k$, then $c$ is a $k$-coloring of $G$.

A valid coloring

An invalid coloring (but still a coloring)
Graph Coloring

The **chromatic number** of a graph $G$ (denoted $\chi(G)$) is the smallest $k$ such that there is a valid *$k$-coloring* of $G$.

**Theorem**

Let $G$ be an undirected graph. Let $\Delta(G)$ be the maximum degree of any vertex in $G$. Then $\chi(G) \leq \Delta(G) + 1$.
Trees

A **tree** is an undirected graph that is *connected* and has *no cycles*

A **free tree** has no organization of vertices and edges.

A **rooted tree** has a designated root at the top (ie the "/" and "a" vertices →).

A free tree can be made rooted by choosing a root.
Rooted Tree Terminology

**Depth of a vertex:** Distance from the root to that vertex

**Height of a tree:** maximum depth of any vertex

**Parent of a vertex:** the node above that vertex (towards the root)
- When $u$ is the parent of $v$, then $v$ is a **child** of $u$
- Vertices with the same parent are called **siblings**

**Leaf:** a vertex with no children
The **ancestors** of a vertex (other than the root) are the vertices in the path from the root to the vertex (including the root, but excluding the vertex). The root has no ancestors.

The **descendants** of a vertex, $v$, are all vertices that have $v$ as an ancestor.

If $a$ is a vertex in a tree, the **subtree rooted at $a$** is the subgraph of the tree consisting of $a$, all of its descendants, and all edges incident to those descendants.
Finite State Machines
[13.2-13.4]
A (deterministic) finite automaton $M$ is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- The set of states $Q$ is finite and non-empty
- The input alphabet $\Sigma$ is finite and non-empty
- The transition function $\delta: Q \times \Sigma \rightarrow Q$
- The starting state $q_0 \in Q$
- The set of final states $F$
Let this automata be $M_1 = (Q_1, \Sigma_1, \delta_1, S_1, F_1)$

$Q_1 = \{ A, B, C, D, INVALID \}$

$\Sigma_1 = \{ $3, $5, $10 \}$

$S_1 = A$

$F_1 = \{ D \}$

$\delta_1 = \{ ((A, $3), INVALID), ((A, $5), B), \ldots \}$
A string $x$ is **recognized** or **accepted** by the machine $M$ if it takes the starting state $q_0$ to a final state.

The **language** that is **recognized** or **accepted** by the machine $M$, denoted by $L(M)$ is the set of strings recognized by $M$.

$$L(M) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}$$
A regular expression, or regex, $r$ over alphabet $\Sigma = \{ c_1, c_2, \ldots, c_k \}$ is:

- $r = c_i$ for some $i \in \{1, \ldots, k\}$
- $r = \emptyset$
- $r = \lambda$

or, given regular expressions $r_1$ and $r_2$, we can build up a new regex $r$:

- $r = (r_1 \mid r_2) \leftarrow r_1 \text{ OR } r_2$, also sometimes written $(r_1 \cup r_2)$
- $r = (r_1r_2) \leftarrow r_1 \text{ concatenated with } r_2$
- $r = (r_1)^* \leftarrow \text{kleene closure (0 or more repetitions)}$