# CSI2101 Discrete Structures: Introduction

Lucia Moura

Winter 2010

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking
  6. applied thinking

# Discrete Mathematics is essential to Computer Science!

- The various aspects of discrete mathematics form the foundation for:
  - modelling computing structures
  - designing programs and algorithms
  - reasoning about programs and algorithms
  - solving real-world problems using the computer
  - in particular, solving problems in information technology, i.e. problems involving computers, communication, information.
- Aspects of discrete mathematics: (David J. Hunter, *Essential of Discrete Mathematics*, 2009)
  1. logical thinking
  2. relational thinking
  3. recursive thinking
  4. quantitative thinking (counting)
  5. analytical thinking
  6. applied thinking
- Question: How these 5 aspects appear in the the activities listed above?

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )
- propositional logic
  propositional calculus allows us to make logical deductions formally

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)
- logic in programming
  imperative programming: conditional statements (if-then-else, do-while)
  logic programming languages (e.g. prolog): uses the rules of predicate logic

# Logical Thinking

- formal logic
  (symbolic manipulation of notation; logical not-thinking :-)))) )
- propositional logic
  propositional calculus allows us to make logical deductions formally
- predicate logic
  make a proposition to depend on a variable and we get a predicate; here the logical deductions include quantifiers (for all, there exists) in front of the predicates)
- methods of proof: direct, by contraposition, by contradiction
  use what you learned in formal/symbolic logic, to guide your reasoning on mathematical proofs (written in paragraph form)
- logic in programming
  imperative programming: conditional statements (if-then-else, do-while)
  logic programming languages (e.g. prolog): uses the rules of predicate logic
- logic in circuits

# Relational Thinking

- It deals with the following type of structures:

# Relational Thinking

- It deals with the following type of structures:
  - sets

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=$n$-ary tuple

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=$n$-ary tuple
  - Dependency of task executions (partial ordering); topological sorting: order tasks respecting dependencies.

# Relational Thinking

- It deals with the following type of structures:
  - sets
  - functions
  - **relations**
  - **partial orderings**
  - **graph theory**
- Question 1: what are each of these structures?
- Question 2: give examples of situations where they can be applied in computer science.
  - Databases: table=relation; record=$n$-ary tuple
  - Dependency of task executions (partial ordering); topological sorting: order tasks respecting dependencies.
  - Graphs: networks (communication, roads, social), conflicts (timetabling, coloring maps), hierarquies (rooted trees), diagrams (binary relations).

# Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.

# Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals

# Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$

# Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$
- Recursive data structures
  e.g. binary search trees

# Recursive Thinking

- Recurrence relations
  Recursively defined sequences of numbers. e.g. Fibonacci sequence.
- Recursive definitions
  e.g. binary trees, recursive geometry/fractals
- Proofs by induction
  Prove that $P(n)$ is true for all $n \geq 0$:
  basis: $P(0)$ is true + induction step $P(n) \Rightarrow P(n+1)$
- Recursive data structures
  e.g. binary search trees
- Recursive algorithms
  e.g. binary search, mergesort, solving towers of Hanoi.

# Quantitative Thinking

- counting,

# Quantitative Thinking

- counting,
- combinations, permutations, arrangements,

# Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,

# Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**

# Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**
- **counting operations in algorithms,**

# Quantitative Thinking

- counting,
- combinations, permutations, arrangements,
- the pigeonhole principle,
- **discrete probability,**
- **counting operations in algorithms,**
- **estimating growth of functions, big-Oh notation.**

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.
  - Writing algorithms that are efficient.

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.
  - Writing algorithms that are efficient.
    - ★ algorithm complexity,

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.
  - Writing algorithms that are efficient.
    - ★ algorithm complexity,
    - ★ analysis of algorithms.

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.
  - Writing algorithms that are efficient.
    - ★ algorithm complexity,
    - ★ analysis of algorithms.
- Question: How previous tools can be applied in each of the above areas?

# Analytical Thinking

- Apply previous tools to analyze problems of interest such as:
  - Writing programs that are correct.
    software engineering tools: testing versus verification.
    - ★ loop invariants,
    - ★ program correctness and verification.
  - Writing algorithms that are efficient.
    - ★ algorithm complexity,
    - ★ analysis of algorithms.
- Question: How previous tools can be applied in each of the above areas?
  - This question will be answered more fully by the studies in this course.

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.
- A lot of the course will focus on acquiring the mathematical skills. But we don't want to lose sight of their use in applications.

# Applied Thinking

- Making the bridge between the mathematical tools and problems we need to solve.
  - problem solving skills
  - modelling
- Before using tools we need to learn the language and methods.
- A lot of the course will focus on acquiring the mathematical skills. But we don't want to lose sight of their use in applications.
- Here we discuss some application problems illustrated in the following slides by Prof. Zaguia (2008): IntroZaguia2008.pdf

# Calendar description:

CSI2101 Discrete Structures (3,1.5,0) 3 cr. Discrete structures as they apply to computer science, algorithm analysis and design. Predicate logic. Review of proof techniques; application of induction to computing problems. Graph theory applications in information technology. Program correctness, preconditions, postconditions and invariants. Analysis of recursive programs using recurrence relations. Properties of integers and basic cryptographical applications. Prerequisite: MAT1348.

# Objectives:

- Discrete mathematics form the foundation for computer science; it is essential in every branch of computing.
- In MAT1348 (discrete mathematics for computing) you have been introduced to fundamental problems and objects in discrete mathematics.
- In CSI2101 (discrete structures) you will learn:
  - more advanced concepts in discrete mathematics
  - more problem solving, modelling, logical reasoning and writing precise proofs
  - how to apply concepts to various types of problems in computing: analyse an algorithm, prove the correctness of a program, model a network problem with graphs, use number theory in cryptography, etc.

# Textbook

References:

- Kenneth H. Rosen, Discrete Mathematics and Its Applications, Sixth Edition, McGraw Hill, 2007.
  (same textbook as normally used for MAT1348; we will use different sections!)

# Topic by topic outline: (approximate number of lectures, order may vary)

1. Introduction (1)
2. Propositionl logic (2)
3. Predicate logic (2)
4. Rules of inference/proof methods (2)
5. Basic number theory and applications (4)
6. Induction and applications. (4)
   Program correctness and verification (1)
7. Solving recurrence relations. Complexity of divide-and-conquer algorithms. (3)
8. Graphs (3)

# Propositional Logic

Lucia Moura

Winter 2010

# Proposition

**A proposition is a declarative sentence that is either true or false.**

Which ones of the following sentences are propositions?

- Ottawa is the capital of Canada.
- Buenos Aires is the capital of Brazil.
- $2 + 2 = 4$
- $2 + 2 = 5$
- if it rains, we don't need to bring an umbrella.
- $x + 2 = 4$
- $x + y = z$
- When does the bus come?
- Do the right thing.

# Propositional variable and connectives

We use letters $p, q, r, \ldots$ to denote **propositional variables** (variables that represent propositions).

We can form new propositions from existing propositions using **logical operators** or **connectives**. These new propositions are called **compound propositions**.

Summary of connectives:

| name | nickname | symbol |
|---|---|---|
| negation | NOT | $\neg$ |
| conjunction | AND | $\wedge$ |
| disjunction | OR | $\vee$ |
| exclusive-OR | XOR | $\oplus$ |
| implication | implies | $\rightarrow$ |
| biconditional | if and only if | $\leftrightarrow$ |

# Meaning of connectives

| $p$ | $q$ | $\neg p$ | $p \wedge q$ | $p \vee q$ | $p \oplus q$ | $p \rightarrow q$ | $p \leftrightarrow q$ |
|---|---|---|---|---|---|---|---|
| T | T | F | T | T | F | T | T |
| T | F | F | F | T | T | F | F |
| F | T | T | F | T | T | T | F |
| F | F | T | F | F | F | T | T |

WARNING:
Implication $(p \rightarrow q)$ causes confusion, specially in line 3: "F $\rightarrow$ T" is true.
One way to remember is that the rule to be obeyed is
"if the premise $p$ is true then the consequence $q$ must be true."
The only truth assignment that falsifies this is $p = T$ and $q = F$.

# Truth tables for compound propositions

Construct the truth table for the compound proposition:
$(p \vee \neg q) \rightarrow (p \wedge q)$

| $p$ | $q$ | $\neg q$ | $p \vee \neg q$ | $p \wedge q$ | $(p \vee \neg q) \rightarrow (p \wedge q)$ |
|---|---|---|---|---|---|
| T | T | F | | | |
| T | F | T | | | |
| F | T | F | | | |
| F | F | T | | | |

# Propositional Equivalences

**A basic step is math is to replace a statement with another with the same truth value (equivalent).**
**This is also useful in order to reason about sentences.**
Negate the following phrase:

"Miguel has a cell phone and he has a laptop computer."

- p="Miguel has a cell phone"
  q="Miguel has a laptop computer."
- The phrase above is written as $(p \wedge q)$.
- Its negation is $\neg(p \wedge q)$, which is logically equivalent to $\neg p \vee \neg q$. (De Morgan's law)
- This negation therefore translates to:
  "Miguel does not have a cell phone or he does not have a laptop computer."

# Truth assignments, tautologies and satisfiability

## Definition

Let $X$ be a set of propositions.
A **truth assignment** (to $X$) is a function $\tau : X \rightarrow \{true, false\}$ that assigns to each propositional variable a truth value. (A truth assignment corresponds to one row of the truth table)
If the truth value of a compound proposition under truth assignment $\tau$ is $true$, we say that $\tau$ **satisfies** $P$, otherwise we say that $\tau$ **falsifies** $P$.

- A compound proposition $P$ is a **tautology** if every truth assignment satisfies $P$, i.e. all entries of its truth table are $true$.
- A compound proposition $P$ is **satisfiable** if there is a truth assignment that satisfies $P$; that is, at least one entry of its truth table is true.
- A compound proposition $P$ is **unsatisfiable (or a contradiction)** if it is not satisfiable; that is, all entries of its truth table are false.

# Examples: tautology, satisfiable, unsatisfiable

For each of the following compound propositions determine if it is a tautology, satisfiable or unsatisfiable:

- $(p \vee q) \wedge \neg p \wedge \neg q$
- $p \vee q \vee r \vee (\neg p \wedge \neg q \wedge \neg r)$
- $(p \rightarrow q) \leftrightarrow (\neg p \vee q)$

# Logical implication and logical equivalence

## Definition

A compound proposition $p$ **logically implies** a compound proposition $q$ (denoted $p \Rightarrow q$) if $p \rightarrow q$ is a tautology.
Two compound propositions $p$ and $q$ are **logically equivalent** (denoted $p \equiv q$, or $p \Leftrightarrow q$ ) if $p \leftrightarrow q$ is a tautology.

## Theorem

*Two compound propositions $p$ and $q$ are logically equivalent if and only if $p$ logically implies $q$ and $q$ logically implies $p$.*

In other words: two compound propositions are logically equivalent if and only if they have the same truth table.

# Logically equivalent compound propositions

Using truth tables to prove that $(p \rightarrow q)$ and $\neg p \vee q$ are logically equivalent, i.e.

$$(p \rightarrow q) \equiv \neg p \vee q$$

| $p$ | $q$ | $\neg p$ | $\neg p \vee q$ | $p \rightarrow q$ |
|---|---|---|---|---|
| T | T | F | T | T |
| T | F | F | F | F |
| F | T | T | T | T |
| F | F | T | T | T |

What is the problem with this approach?

# Truth tables versus logical equivalences

Truth tables grow exponentially with the number of propositional variables!

A truth table with $n$ variables has $2^n$ rows.

Truth tables are practical for small number of variables, but if you have, say, 7 variables, the truth table would have 128 rows!

Instead, we can prove that two compound propositions are logically equivalent by using known logical equivalences ("equivalence laws").

# Summary of important logical equivalences I

**TABLE 6 Logical Equivalences.**

| *Equivalence* | *Name* |
|---|---|
| $p \wedge \mathbf{T} \equiv p$<br>$p \vee \mathbf{F} \equiv p$ | Identity laws |
| $p \vee \mathbf{T} \equiv \mathbf{T}$<br>$p \wedge \mathbf{F} \equiv \mathbf{F}$ | Domination laws |
| $p \vee p \equiv p$<br>$p \wedge p \equiv p$ | Idempotent laws |
| $\neg(\neg p) \equiv p$ | Double negation law |
| $p \vee q \equiv q \vee p$<br>$p \wedge q \equiv q \wedge p$ | Commutative laws |
| $(p \vee q) \vee r \equiv p \vee (q \vee r)$<br>$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ | Associative laws |
| $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$<br>$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ | Distributive laws |
| $\neg(p \wedge q) \equiv \neg p \vee \neg q$<br>$\neg(p \vee q) \equiv \neg p \wedge \neg q$ | De Morgan's laws |
| $p \vee (p \wedge q) \equiv p$<br>$p \wedge (p \vee q) \equiv p$ | Absorption laws |
| $p \vee \neg p \equiv \mathbf{T}$<br>$p \wedge \neg p \equiv \mathbf{F}$ | Negation laws |

Note T is the compound composition that is always true, and F is the compound composition that is always false.

# Summary of important logical equivalences II

**TABLE 7 Logical Equivalences Involving Conditional Statements.**

$$p \to q \equiv \neg p \vee q$$

$$p \to q \equiv \neg q \to \neg p$$

$$p \vee q \equiv \neg p \to q$$

$$p \wedge q \equiv \neg(p \to \neg q)$$

$$\neg(p \to q) \equiv p \wedge \neg q$$

$$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$$

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

$$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$$

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

**TABLE 8 Logical Equivalences Involving Biconditionals.**

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$$

$$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$$

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

$$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$$

Rosen, page 24-25.

# Proving new logical equivalences

Use known logical equivalences to prove the following:

1. Prove that $\neg(p \rightarrow q) \equiv p \wedge \neg q$.

2. Prove that $(p \wedge q) \rightarrow (p \vee q)$ is a tautology.

# Normal forms for compound propositions

- A literal is a propositional variable or the negation of a propositional variable.
- A term is a literal or the conjunction (and) of two or more literals.
- A clause is a literal or the disjunction (or) of two or more literals.

## Definition

A compound proposition is in **disjunctive normal form** (DNF) if it is a term or a disjunction of two or more terms. (i.e. an OR of ANDs).
A compound proposition is in **conjunctive normal form** (CNF) if it is a clause or a conjunction of two or more clauses. (i.e. and AND of ORs)

# Disjunctive normal form (DNF)

| | $x$ | $y$ | $z$ | $x \vee y \rightarrow \neg x \wedge z$ |
|---|---|---|---|---|
| 1 | F | F | F | T |
| 2 | F | F | T | T |
| 3 | F | T | F | F |
| 4 | F | T | T | T |
| 5 | T | F | F | F |
| 6 | T | F | T | F |
| 7 | T | T | F | F |
| 8 | T | T | T | F |

The formula is satisfied by the truth assignment in row 1 **or** by the truth assignment in row 2 **or** by the truth assignment in row 4.
So, its DNF is : $(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge z)$

# Conjunctive normal form (CNF)

| | $x$ | $y$ | $z$ | $x \vee y \rightarrow \neg x \wedge z$ |
|---|---|---|---|---|
| 1 | F | F | F | T |
| 2 | F | F | T | T |
| 3 | F | T | F | F |
| 4 | F | T | T | T |
| 5 | T | F | F | F |
| 6 | T | F | T | F |
| 7 | T | T | F | F |
| 8 | T | T | T | F |

The formula is **not** satisfied by the truth assignment in row 3 **and** in row 5 **and** in row 6 **and** in row 7 **and** in row 8. So:, it is log. equiv. to:

$\neg(\neg x \wedge y \wedge \neg z) \wedge \neg(x \wedge \neg y \wedge \neg z) \wedge \neg(x \wedge \neg y \wedge z) \wedge \neg(x \wedge y \wedge \neg z) \wedge \neg(x \vee y \vee z)$

apply DeMorgan's law to obtain its CNF:

$(x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \wedge \neg y \wedge \neg z)$

# Boolean functions and the design of digital circuits

Let $B = \{false, true\}$ (or $B = \{0, 1\}$). A function $f : B^n \to B$ is called a boolean function of degree $n$.

## Definition

A compound proposition $P$ with propositions $x_1, x_2, \ldots, x_n$ represents a Boolean function $f$ with arguments $x_1, x_2, \ldots, x_n$ if for any truth assignment $\tau$, $\tau$ satisfies $P$ if and only if $f(\tau(x_1), \tau(x_2), \ldots, \tau(x_n)) = true$.

## Theorem

*Let $P$ be a compound proposition that represents a boolean function $f$. Then, a compound proposition $Q$ also represents $f$ if and only if $Q$ is logically equivalent to $P$.*

# Complete set of connectives (functionally complete)

## Theorem

*Every boolean formula can be represented by a compound proposition that uses only connectives* $\{\neg, \wedge, \vee\}$ *(i.e.* $\{\neg, \wedge, \vee\}$ *is* **functionally complete** *).*

**Proof: use DNF or CNF!**

This is the basis of circuit design:

In digital circuit design, we are given a **functional specification** of the circuit and we need to construct a **hardware implementation**.

**functional specification** = number $n$ of inputs + number $m$ of outputs + describe outputs for each set of inputs (i.e. $m$ boolean functions!)

**Hardware implementation** uses logical gates: or-gates, and-gates, inverters.

The functional specification corresponds to $m$ boolean functions which we can represent by $m$ compound propositions that uses only $\{\neg, \wedge, \vee\}$, that is, its hardware implementation uses inverters, and-gates and or-gates.

# Boolean functions and digital circuits

Consider the boolean function represented by $x \vee y \rightarrow \neg x \wedge z$.

Give a digital circuit that computes it, using only $\{\wedge, \vee, \neg\}$.
This is always possible since $\{\wedge, \vee, \neg\}$ is functionally complete (e.g. use DNF or CNF).

Give a digital circuit that computes it, using only $\{\wedge, \neg\}$.
This is always possible, since $\{\wedge, \neg\}$ is **functionally complete**:
Proof: Since $\{\wedge, \vee, \neg\}$ is functionally complete, it is enough to show how to express $x \vee y$ using only $\{\wedge, \neg\}$:
$(x \vee y) \equiv \neg(\neg x \wedge \neg y)$

Give a digital circuit that computes it, using only $\{\vee, \neg\}$.
Prove that $\{\vee, \neg\}$ is **functionally complete**.

# Predicate Logic

Lucia Moura

Winter 2010

# Predicates

A Predicate is a declarative sentence whose true/false value depends on one or more variables.
The statement "x is greater than 3" has two parts:

- the subject: $x$ is the subject of the statement
- the predicate: "is greater than 3" (a property that the subject can have).

We denote the statement "$x$ is greater than 3" by $P(x)$, where $P$ is the predicate "is greater than 3" and $x$ is the variable.
The statement $P(x)$ is also called the value of **propositional function** $P$ at $x$.
Assign a value to $x$, so $P(x)$ becomes a proposition and has a truth value:
$P(5)$ is the statement "5 is greater than 3", so $P(5)$ is true.
$P(2)$ is the statement "2 is greater than 3", so $P(2)$ is false.

# Predicates: Examples

Given each propositional function determine its true/false value when variables are set as below.

- Prime$(x)$ = "$x$ is a prime number."
  Prime$(2)$ is true, since the only numbers that divide $2$ are $1$ and itself.
  Prime$(9)$ is false, since $3$ divides $9$.
- $C(x, y)$="$x$ is the capital of $y$".
  $C$(Ottawa,Canada) is true.
  $C$(Buenos Aires,Brazil) is false.
- $E(x, y, z)$ = "$x + y = z$".
  $E(2, 3, 5)$ is ...
  $E(4, 4, 17)$ is ...

# Quantifiers

Assign a value to $x$ in $P(x) =$"$x$ is an odd number", so the resulting statement becomes a proposition: $P(7)$ is true, $P(2)$ is false.

**Quantification** is another way to create propositions from a propositional functions:

- **universal** quantification: $\forall x P(x)$ says
  "the predicate P is true for every element under consideration."
  Under the domain of natural numbers, $\forall x P(x)$ is false.
- **existencial** quantification: $\exists x P(x)$ says
  "there is one or more element under consideration for which the predicate $P$ is true."
  Under the domain of natural numbers, $\exists x P(x)$ is true, since for instance $P(7)$ is true.

**Predicate calculus:** area of logic dealing with predicates and quantifiers.

# Domain, domain of discourse, universe of discourse

Before deciding on the truth value of a quantified predicate, it is mandatory to specify the **domain** (also called domain of discourse or universe of discourse).

$P(x) =$ "$x$ is an odd number"

$\forall x P(x)$ is **false** for the domain of **integer numbers**; but
$\forall x P(x)$ is **true** for the domain of **prime numbers greater than 2**.

# Universal Quantifier

The **universal quantification** of $P(x)$ is the statement:
"$P(x)$ for all values of $x$ in the domain" denoted $\forall x P(x)$.

$\forall x P(x)$ is **true** when $P(x)$ is true for every $x$ in the domain.

$\forall x P(x)$ is **false** when there is an $x$ for which $P(x)$ is false.
An element for which $P(x)$ is false is called a **counterexample** of $\forall x P(x)$.

If the domain is empty, $\forall x P(x)$ is true for any propositional function $P(x)$, since there are no counterexamples in the domain.

If the domain is finite $\{x_1, x_2, \ldots, x_n\}$, $\forall x P(x)$ is the same as

$$P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n).$$

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - False. $3$ is a counterexample.

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - False. $3$ is a counterexample.
  - Also note that here $\forall P(x)$ is $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, so its enough to observe that $P(3)$ is false.

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - False. $3$ is a counterexample.
  - Also note that here $\forall P(x)$ is $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, so its enough to observe that $P(3)$ is false.
  - the set of real numbers in the interval $[10, 39.5]$

# Universal quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\forall x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - False. $3$ is a counterexample.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - False. $3$ is a counterexample.
  - Also note that here $\forall P(x)$ is $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, so its enough to observe that $P(3)$ is false.
  - the set of real numbers in the interval $[10, 39.5]$
  - True. It takes a bit longer to verify than in false statements.
    Let $x \in [10, 39.5]$. Then $x \geq 10$ which implies $x^2 \geq 10^2 = 100 > 10$, and so $x^2 > 10$.

# Existencial Quantifier

The **existential quantification** of $P(x)$ is the statement:
"There exists an element $x$ in the domain such that $P(x)$" denoted $\exists x P(x)$.

$\exists x P(x)$ is **true** when $P(x)$ is true for one or more $x$ in the domain.
An element for which $P(x)$ is true is called a **witness** of $\exists x P(x)$.

$\exists x P(x)$ is **false** when $P(x)$ is false for every $x$ in the domain (if domain nonempty).

If the domain is empty, $\exists x P(x)$ is false for any propositional function $P(x)$, since there are no witnesses in the domain.

If the domain is finite $\{x_1, x_2, \ldots, x_n\}$, $\exists x P(x)$ is the same as

$$P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n).$$

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - True. $4$ is a witness.

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - True. $4$ is a witness.
  - Also note that here $\exists P(x)$ is $P(1) \vee P(2) \vee P(3) \vee P(4)$, so its enough to observe that $P(4)$ is true.

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - True. $4$ is a witness.
  - Also note that here $\exists P(x)$ is $P(1) \vee P(2) \vee P(3) \vee P(4)$, so its enough to observe that $P(4)$ is true.
  - the set of real numbers in the interval $[0, \sqrt{9.8}]$

# Existencial quantifiers: example

- Let $P(x)$ be "$x^2 > 10$".
  What is the truth value of $\exists x P(x)$ for each of the following domains:
  - the set of real numbers: $\mathbb{R}$
  - True. $10$ is a witness.
  - the set of positive integers not exceeding 4: $\{1, 2, 3, 4\}$
  - True. $4$ is a witness.
  - Also note that here $\exists P(x)$ is $P(1) \vee P(2) \vee P(3) \vee P(4)$, so its enough to observe that $P(4)$ is true.
  - the set of real numbers in the interval $[0, \sqrt{9.8}]$
  - False. It takes a bit longer to conclude than in true statements.
    Let $x \in [0, 9.8]$. Then $0 \leq x \leq \sqrt{9.8}$ which implies
    $x^2 \leq (\sqrt{9.8})^2 = 9.8 < 10$, and so $x^2 < 10$.
    What we have shown is that $\forall x \neg P(x)$, which (we will see) is equivalent to $\neg \exists x P(x)$

# Other forms of quantification

- **Other Quantifiers**
  The most important quantifiers are $\forall$ and $\exists$, but we could define many different quantifiers: "there is a unique", "there are exactly two", "there are no more than three", "there are at least 100", etc.
  A common one is the **uniqueness quantifier**, denoted by $\exists!$.
  $\exists! x P(x)$ states "There exists a unique $x$ such that $P(x)$ is true."
  Advice: stick to the basic quantifiers. We can write $\exists! x P(x)$ as $\exists x(P(x) \wedge \forall y(P(y) \rightarrow y = x))$ or more compactly $\exists x \forall y(P(y) \leftrightarrow y = x)$
- **Restricting the domain of a quantifier**
  Abbreviated notation is allowed, in order to restrict the domain of certain quantifiers.
  - $\forall x > 0(x^2 > 0)$ is the same as $\forall x(x > 0 \rightarrow x^2 > 0)$.
  - $\forall y \neq 0(y^3 \neq 0)$ is the same as $\forall y(y \neq 0 \rightarrow y^3 \neq 0)$.
  - $\exists z > 0(z^2 = 2)$ is the same as $\exists x(z > 0 \wedge z^2 = 2)$

# Precedence and scope of quantifiers

- $\forall$ and $\exists$ have higher precedence than logical operators.
  Example: $\forall x P(x) \vee Q(x)$ means $(\forall x P(x)) \vee Q(x)$, it doesn't mean $\forall x(P(x) \vee Q(x))$.
  (Note: This statement is not a proposition since there is a free variable!)
- **Binding variables and scope**
  When a quantifier is used on the variable $x$ we say that this occurrence of $x$ is **bound**. When the occurrence of a variable is not bound by a quantifier or set to a particular value, the variable is said to be **free**.
  The part of a logical expression to which a quantifier is applied is the **scope** of the quantifier. A variable is free if it is outside the scope of all quantifiers.
  In the example above, $(\forall x \underline{P(x)}) \vee Q(x)$, the $x$ in $P(x)$ is bound by the existencial quantifier, while the $x$ in $Q(x)$ is free. The scope of the universal quantifier is underlined.

# Logical Equivalences Involving Quantifiers

## Definition

Two statements $S$ and $T$ involving predicates and quantifiers are *logically equivalent* if and only if they have the same truth value regardless of the **interpretation**, i.e. regardless of

- the meaning that is attributed to each propositional function,
- the domain of discourse.

We denote $S \equiv T$.

Is $\forall x(P(x) \wedge Q(x))$ **logically equivalent** to $\forall x P(x) \wedge \forall x Q(x)$ ?
Is $\forall x(P(x) \vee Q(x))$ **logically equivalent** to $\forall x P(x) \vee \forall x Q(x)$ ?

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).
- Use two steps:

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).
- Use two steps:
  - If $\forall x(P(x) \wedge Q(x))$ is true, then $\forall x P(x) \wedge \forall x Q(x)$ is true.

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).
- Use two steps:
  - If $\forall x(P(x) \wedge Q(x))$ is true, then $\forall x P(x) \wedge \forall x Q(x)$ is true.
  - Proof: Suppose $\forall x(P(x) \wedge Q(x))$ is true.
    Then if $a$ is in the domain, $P(a) \wedge Q(a)$ is true, and so $P(a)$ is true and $Q(a)$ is true.
    So, if $a$ in in the domain $P(a)$ is true, which is the same as $\forall x P(x)$ is true; and similarly, we get that $\forall x Q(x)$ is true.
    This means that $\forall x P(x) \wedge \forall x Q(x)$ is true.

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).
- Use two steps:
  - If $\forall x(P(x) \wedge Q(x))$ is true, then $\forall x P(x) \wedge \forall x Q(x)$ is true.
  - Proof: Suppose $\forall x(P(x) \wedge Q(x))$ is true.
    Then if $a$ is in the domain, $P(a) \wedge Q(a)$ is true, and so $P(a)$ is true and $Q(a)$ is true.
    So, if $a$ in in the domain $P(a)$ is true, which is the same as $\forall x P(x)$ is true; and similarly, we get that $\forall x Q(x)$ is true.
    This means that $\forall x P(x) \wedge \forall x Q(x)$ is true.
  - If $\forall x P(x) \wedge \forall x Q(x)$ is true, then $\forall x(P(x) \wedge Q(x))$ is true.

- Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall x P(x) \wedge \forall x Q(x)$ (where the same domain is used throughout).
- Use two steps:
  - If $\forall x(P(x) \wedge Q(x))$ is true, then $\forall x P(x) \wedge \forall x Q(x)$ is true.
  - Proof: Suppose $\forall x(P(x) \wedge Q(x))$ is true.
    Then if $a$ is in the domain, $P(a) \wedge Q(a)$ is true, and so $P(a)$ is true and $Q(a)$ is true.
    So, if $a$ in in the domain $P(a)$ is true, which is the same as $\forall x P(x)$ is true; and similarly, we get that $\forall x Q(x)$ is true.
    This means that $\forall x P(x) \wedge \forall x Q(x)$ is true.
  - If $\forall x P(x) \wedge \forall x Q(x)$ is true, then $\forall x(P(x) \wedge Q(x))$ is true.
  - Proof: Suppose that $\forall x P(x) \wedge \forall x Q(x)$ is true.
    It follows that $\forall x P(x)$ is true and $\forall x Q(x)$ is true.
    So, if $a$ is in the domain, then $P(a)$ is true and $Q(a)$ is true.
    It follows that if $a$ is in the domain $P(a) \wedge Q(a)$ is true.
    This means that $\forall x(P(x) \wedge Q(x))$ is true.

- Prove that $\forall x(P(x) \vee Q(x))$ **is not logically equivalent** to $\forall x P(x) \vee \forall x Q(x)$.

- Prove that $\forall x(P(x) \vee Q(x))$ **is not logically equivalent** to $\forall x P(x) \vee \forall x Q(x)$.
- It is enough to give a counterexample to the assertion that they have the same truth value for all possible interpretations.

- Prove that $\forall x(P(x) \vee Q(x))$ **is not logically equivalent** to $\forall x P(x) \vee \forall x Q(x)$.
- It is enough to give a counterexample to the assertion that they have the same truth value for all possible interpretations.
- Under the following interpretation:
  domain: set of people in the world
  $P(x) =$"$x$ is male".
  $Q(x) =$"$x$ is female".

  We have:
  $\forall x(P(x) \vee Q(x))$ (every person is a male or a female) is true;
  while $\forall x P(x) \vee \forall x Q(x)$ (every person is a male or every person is a female) is false.

# Negating Quantified Expressions: De Morgan Laws

$$\neg\forall x P(x) \equiv \exists x \neg P(x)$$

Proof:
$\neg\forall x P(x)$ is true if and only if $\forall x P(x)$ is false.
Note that $\forall x P(x)$ is false if and only if there exists an element in the domain for which $P(x)$ is false.
But this holds if and only if there exists an element in the domain for which $\neg P(x)$ is true.
The latter holds if and only if $\exists x \neg P(x)$ is true.

# De Morgan Laws for quantifiers (continued)

$$\neg\exists x P(x) \equiv \forall x \neg P(x)$$

Proof:
$\neg\exists P(x)$ is true if and only if $\exists x P(x)$ is false.
Note that $\exists x P(x)$ is false if and only there exists no element in the domain for which $P(x)$ is true.
But this holds if and only if for all elements in the domain we have $P(x)$ is false;
which is the same as for all elements in the domain we have $\neg P(x)$ is true.
The latter holds if and only if $\forall x \neg P(x)$ is true.

# Practice Exercises

1. What are the negatons of the following statements:
   "There is an honest politician."
   "All americans eat cheeseburgers."
2. What are the negations of $\forall x(x^2 > x)$ and $\exists x(x^2 = 2)$?
3. Show that $\neg\forall x(P(x) \rightarrow Q(x))$ and $\exists x(P(x) \wedge \neg Q(x))$ are logically equivalent.

Solutions in the textbook's page 41.

# Example from Lewis Caroll's book *Symbolic Logic*

Consider these statements (two premises followed by a conclusion):
"All lions are fierce."
"Some lions do not drink coffee."
"Some fierce creatures do not drink coffee."

Assume that the domain is the set of all creatures and $P(x) =$"$x$ is a lion", $Q(x) =$"$x$ is fierce", $R(x) =$"$x$ drinks coffee".

Exercise: Express the above statements using $P(x)$, $Q(x)$ and $R(x)$, under the domain of all creatures.

*Is the conclusion a valid consequence of the premises?*
*In this case, yes. (See more on this type of derivation, in a future lecture on Rules of Inference).*

# Nested Quantifiers

Two quantifiers are nested if one is in the scope of the other.
Everything within the scope of a quantifier can be thought of as a propositional function.
For instance,

"$\forall x \exists y(x + y = 0)$" is the same as
"$\forall x Q(x)$", where $Q(x)$ is "$\exists y(x + y = 0)$".

# The order of quantifiers

Let $P(x, y)$ be the statement "$x + y = y + x$".
Consider the following:
$\forall x \forall y P(x, y)$ and $\forall y \forall x P(x, y)$.

- What is the meaning of each of these statements?
- What is the truth value of each of these statements?
- Are they equivalent?

Let $Q(x, y)$ be the statement "$x + y = 0$".
Consider the following:
$\exists y \forall x Q(x, y)$ and $\forall x \exists y Q(x, y)$.

- What is the meaning of each of these statements?
- What is the truth value of each of these statements?
- Are they equivalent?

# Summary of quantification of two variables

| statement | when true ? | when false ? |
|---|---|---|
| $\forall x \forall y P(x,y)$<br>$\forall y \forall x P(x,y)$ | $P(x,y)$ is true<br>for every pair $x$, $y$. | There is a pair $x, y$ for<br>which $P(x,y)$ is false |
| $\forall x \exists y P(x,y)$ | For every $x$ there is $y$<br>for which $P(x,y)$ is true | There is an $x$ such that<br>$P(x,y)$ is false for every $y$ |
| $\exists x \forall y P(x,y)$ | There is an $x$ for which<br>$P(x,y)$ is true for every $y$ | For every $x$ there is a $y$<br>for which $P(x,y)$ is false. |
| $\exists x \exists y P(x,y)$<br>$\exists y \exists x P(x,y)$ | There is a pair $x$, $y$<br>for which $P(x,y)$ is true | $P(x,y)$ is false<br>for every pair $x$, $y$. |

# Translating Math Statements into Nested quantifiers

Translate the following statements:

1. "The sum of two positive integers is always positive."
2. "Every real number except zero has a multiplicative inverse." (a multiplicative inverse of $x$ is $y$ such that $xy = 1$).
3. "Every positive integer is the sum of the squares of four integers."

# Translating from Nested Quantifiers into English

Let $C(x)$ denote "$x$ has a computer" and $F(x, y)$ be "$x$ and $y$ are friends.", and the domain be all students in your school.
Translate:

1. $\forall x(C(x) \vee \exists y(C(y) \wedge F(x, y))$.
2. $\exists x \forall y \forall z((F(x, y) \wedge F(x, z) \wedge (y \neq z)) \rightarrow \neg F(y, z))$

# Translating from English into Nested Quantifiers

1. "If a person is female and is a parent, then this person is someone's mother."
2. "Everyone has exactly one best friend."
3. "There is a woman who has taken a flight on every airline of the world."

# Negating Nested Quantifiers

Express the negation of the following statements, so that no negation precedes a quantifier (apply DeMorgan successively):

- $\forall x \exists y (xy = 1)$
- $\forall x \exists y P(x, y) \vee \forall x \exists y Q(x, y)$
- $\forall x \exists y (P(x, y) \wedge \exists z R(x, y, z))$

# Predicate calculus in Mathematical Reasoning

Using predicates to express definitions.
$D(x) =$"$x$ is a prime number"
(defined term)
$P(x) =$"$x \geq 2$ and the only divisors of $x$ are $1$ and $x$"
(defining property about $x$)
Definition of prime number: $\forall x(D(x) \leftrightarrow P(x))$

Note that definitions in English form use *if* instead of *if and only if*, but we really mean *if and only if*.

# Predicate calculus in Mathematical Reasoning (cont'd)

Let $P(n, x, y, z)$ be the predicate $x^n + y^n = z^n$.

1. Write the following statements in predicate logic, using the domain of positive integers:
   "For every integer $n > 2$, there does not exist positive integers $x$, $y$ and $z$ such that $x^n + y^n = z^n$."
2. Negate the previous statement, and simplify it so that no negation precedes a quantifier.
3. What needs to be found in order to give a counter example to 1 ?
4. Which famous theorem is expressed in 1, who proved and when?

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.
- Originally $x = a$ and $y = b$, by $P(x, y)$.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.
- Originally $x = a$ and $y = b$, by $P(x, y)$.
- After step 1, $x = a$, $temp = a$ and $y = b$.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.
- Originally $x = a$ and $y = b$, by $P(x, y)$.
- After step 1, $x = a$, $temp = a$ and $y = b$.
- After step 2, $x = b$, $temp = a$ and $y = b$.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.
- Originally $x = a$ and $y = b$, by $P(x, y)$.
- After step 1, $x = a$, $temp = a$ and $y = b$.
- After step 2, $x = b$, $temp = a$ and $y = b$.
- After step 3, $x = b$, $temp = a$ and $y = a$.

# Predicate calculus in Program Verification: a toy example

The following program is designed to exchange the value of $x$ and $y$:

```
temp := x
x := y
y := temp
```

Find preconditions, postconditions and verify its correctness.

- Precondition: $P(x, y)$ is "$x = a$ and $y = b$", where $a$ and $b$ are the values of $x$ and $y$ before we execute these 3 statements.
- Postconditon: $Q(x, y)$ is "$x = b$ and $y = a$".
- Assume $P(x, y)$ holds before and show that $Q(x, y)$ holds after.
- Originally $x = a$ and $y = b$, by $P(x, y)$.
- After step 1, $x = a$, $temp = a$ and $y = b$.
- After step 2, $x = b$, $temp = a$ and $y = b$.
- After step 3, $x = b$, $temp = a$ and $y = a$.
- Therefore, after the program we know $Q(x, y)$ holds.

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"
   - $\forall m(S(m, 1) \rightarrow C(m))$

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"
   - $\forall m(S(m, 1) \rightarrow C(m))$
2. "If a user is active, at least one network link will be available."

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"
   - $\forall m(S(m, 1) \rightarrow C(m))$
2. "If a user is active, at least one network link will be available."
   - $A(u)$: "User $u$ is active."

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"
   - $\forall m(S(m, 1) \rightarrow C(m))$
2. "If a user is active, at least one network link will be available."
   - $A(u)$: "User $u$ is active."
   - $S(n, x)$: "Network $n$ is in state $x$".

# Predicate calculus in System Specification

Use predicates and quantifiers to express system specifications:

1. "Every mail message larger than one megabyte will be compressed."
   - $S(m, y)$: "mail message $m$ is larger than $y$ megabytes"
   - $C(m)$: "mail message $m$ will be compressed"
   - $\forall m(S(m, 1) \rightarrow C(m))$
2. "If a user is active, at least one network link will be available."
   - $A(u)$: "User $u$ is active."
   - $S(n, x)$: "Network $n$ is in state $x$".
   - $\exists u A(u) \rightarrow \exists n S(n, available)$

# Predicate calculus in Logic Programming

Prolog is a declarative language based in predicate logic.
The program is expressed as **Prolog facts** and **Prolog rules**.
Execution is triggered by running queries over these relations.

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

The result of the following query is given:

```
?- sibling(sally, erica).
Yes
```

# Inference Rules and Proof Methods

Lucia Moura

Winter 2010

# Rules of Inference and Formal Proofs

Proofs in mathematics are valid arguments that establish the truth of mathematical statements.

- An **argument** is a sequence of statements that end with a conclusion.
- The argument is **valid** if the conclusion (final statement) follows from the truth of the preceding statements (**premises**).

Rules of inference are templates for building valid arguments.

We will study rules of inferences for compound propositions, for quantified statements, and then see how to combine them.

These will be the main ingredients needed in **formal proofs**.

# Proof methods and Informal Proofs

After studying how to write **formal proofs** using rules of inference for predicate logic and quantified statements, we will move to **informal proofs**.

Proving useful theorems using **formal proofs** would result in long and tedious proofs, where every single logical step must be provided.

Proofs used for human consumption (rather than for automated derivations by the computer) are usually **informal proofs**, where steps are combined or skipped, axioms or rules of inference are not explicitly provided.

The second part of these slides will cover methods for writing informal proofs.

# Valid Arguments using Propositional Logic

Consider the following argument (sequence of propositions):

- If the prof offers chocolate for an answer, you answer the prof's question.
- The prof offers chocolate for an answer.
- Therefore, you answer the prof's question.

Let $p$ be "the prof offers chocolate for an answer"
and $q$ be "you answer the prof's question".

The *form* of the above argument is:

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

The argument is valid since $((p \rightarrow q) \wedge p) \rightarrow q$ is a tautology.

# Arguments, argument forms and their validity

## Definition

An *argument* in propositional logic is sequence of propositions. All but the final proposition are called *premises* and the final proposition is called the *conclusion*. An argument is *valid* if the truth of all its premises implies that the conclusion is true.
An *argument form* in propositional logic is a sequence of compound propositions involving propositional variables. An argument form is *valid* if no matter which propositions are substituted for the propositional variables in its premises, if the premises are all true, then the conclusion is true.

In other words, an argument form with premises $p_1, p_2, \ldots, p_n$ and conclusion $q$ is valid if and only if

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow q$$

is a tautology.

# Rules of Inference for Propositional Logic I

| inference rule | tautology | name |
|---|---|---|
| $p$<br>$p \rightarrow q$<br>$\therefore \quad q$ | $(p \wedge (p \rightarrow q)) \rightarrow q$ | Modus ponens<br>(mode that affirms) |
| $\neg q$<br>$p \rightarrow q$<br>$\therefore \quad \neg p$ | $(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$ | Modus tollens<br>(mode that denies) |
| $p \rightarrow q$<br>$q \rightarrow r$<br>$\therefore \quad p \rightarrow r$ | $((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$ | hypothetical syllogism |
| $p \vee q$<br>$\neg p$<br>$\therefore \quad q$ | $((p \vee q) \wedge (\neg p)) \rightarrow q$ | disjunctive syllogism |

# Rules of Inference for Propositional Logic II

| | | |
|---|---|---|
| $\therefore \dfrac{p}{p \vee q}$ | $p \rightarrow (p \vee q)$ | addition |
| $\therefore \dfrac{p \wedge q}{p}$ | $(p \wedge q) \rightarrow p$ | simplification |
| $\therefore \dfrac{\begin{array}{c}p \\ q\end{array}}{p \wedge q}$ | $((p) \wedge (q)) \rightarrow (p \wedge q)$ | conjunction |
| $\therefore \dfrac{\begin{array}{c}p \vee q \\ \neg p \vee r\end{array}}{q \vee r}$ | $((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$ | resolution |

# Which rule of inference is used in each argument below?

- Alice is a Math major. Therefore, Alice is either a Math major or a CSI major.
- Jerry is a Math major and a CSI major. Therefore, Jerry is a Math major.
- If it is rainy, then the pool will be closed. It is rainy. Therefore, the pool is closed.
- If it snows today, the university will close. The university is not closed today. Therefore, it did not snow today.
- If I go swimming, then I will stay in the sun too long. If I stay in the sun too long, then I will sunburn. Therefore, if I go swimming, then I will sunburn.
- I go swimming or eat an ice cream. I did not go swimming. Therefore, I eat an ice cream.

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?
- Does the conclusion must be true?

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?
- Does the conclusion must be true?
- What is wrong?

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?
- Does the conclusion must be true?
- What is wrong?
- The argument is valid: modus ponens inference rule.

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?
- Does the conclusion must be true?
- What is wrong?
- The argument is valid: modus ponens inference rule.
- We cannot conclude that the conclusion is true, since one of its premises, $\sqrt{2} > \frac{3}{2}$, is false.

# Determine whether the argument is valid and whether the conclusion must be true

- If $\sqrt{2} > \frac{3}{2}$ then $(\sqrt{2})^2 > (\frac{3}{2})^2$. We know that $\sqrt{2} > \frac{3}{2}$. Therefore, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$.
- Is the argument valid?
- Does the conclusion must be true?
- What is wrong?
- The argument is valid: modus ponens inference rule.
- We cannot conclude that the conclusion is true, since one of its premises, $\sqrt{2} > \frac{3}{2}$, is false.
- Indeed, in this case the conclusion is false, since $2 \not> \frac{9}{4} = 2.25$.

# Formal Proofs: using rules of inference to build arguments

## Definition

A **formal proof** of a conclusion $q$ given hypotheses $p_1, p_2, \ldots, p_n$ is a sequence of steps, each of which applies some inference rule to hypotheses or previously proven statements (antecedents) to yield a new true statement (the consequent).

A formal proof demonstrates that if the premises are true, then the conclusion is true.

*Note that the word* formal *here is not a synomym of rigorous.*

*A formal proof is based simply on symbol manipulation (no need of thinking, just apply rules).*

*A formal proof is rigorous but so can be a proof that does not rely on symbols!*

# Formal proof example

Show that the hypotheses:

- It is not sunny this afternoon and it is colder than yesterday.
- We will go swimming only if it is sunny.
- If we do not go swimming, then we will take a canoe trip.
- If we take a canoe trip, then we will be home by sunset.

lead to the conclusion:

- We will be home by the sunset.

Main steps:

1. Translate the statements into proposional logic.
2. Write a formal proof, a sequence of steps that state hypotheses or apply inference rules to previous steps.

Show that the hypotheses:

- It is not sunny this afternoon and it is colder than yesterday. $\neg s \wedge c$
- We will go swimming only if it is sunny. $w \rightarrow s$
- If we do not go swimming, then we will take a canoe trip. $\neg w \rightarrow t$
- If we take a canoe trip, then we will be home by sunset. $t \rightarrow h$

lead to the conclusion:

- We will be home by the sunset. $h$

| Step | Reason |
|---|---|
| 1. $\neg s \wedge c$ | hypothesis |
| 2. $\neg s$ | simplification |
| 3. $w \rightarrow s$ | hypothesis |
| 4. $\neg w$ | modus tollens of 2 and 3 |
| 5. $\neg w \rightarrow t$ | hypothesis |
| 6. $t$ | modus ponens of 4 and 5 |
| 7. $t \rightarrow h$ | hypothesis |
| 8. $h$ | modus ponens of 6 and 7 |

Where:
$s$: "it is sunny this afternoon"
$c$: "it is colder than yesterday"
$w$: "we will go swimming"
$t$: "we will take a canoe trip.
$h$: "we will be home by the sunset."

# Resolution and Automated Theorem Proving

We can build programs that automate the task of reasoning and proving theorems.

Recall that the rule of inference called **resolution** is based on the tautology:

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

If we express the hypotheses and the conclusion as **clauses** (possible by CNF, a conjunction of clauses), we can use **resolution** as the only inference rule to build proofs!

This is used in programming languages like **Prolog**.
It can be used in **automated theorem proving systems**.

# Proofs that use exclusively **resolution** as inference rule

Step 1: Convert hypotheses and conclusion into clauses:

| Original hypothesis | equivalent CNF | Hypothesis as list of clauses |
|---|---|---|
| $(p \wedge q) \vee r$<br>$r \rightarrow s$ | $(p \vee r) \wedge (q \vee r)$<br>$(\neg r \vee s)$ | $(p \vee r)$, $(q \vee r)$<br>$(\neg r \vee s)$ |

| Conclusion | equivalent CNF | Conclusion as list of clauses |
|---|---|---|
| $p \vee s$ | $(p \vee s)$ | $(p \vee s)$ |

Step 2: Write a proof based on resolution:

| Step | Reason |
|---|---|
| 1. $p \vee r$ | hypothesis |
| 2. $\neg r \vee s$ | hypothesis |
| 3. $p \vee s$ | resolution of 1 and 2 |

Show that the hypotheses:

- $\neg s \wedge c$ translates to clauses: $\neg s$, $c$
- $w \rightarrow s$ translates to clause: $(\neg w \vee s)$
- $\neg w \rightarrow t$ translates to clause: $(w \vee t)$
- $t \rightarrow h$ translates to clause: $(\neg t \vee h)$

lead to the conclusion:

- $h$ (it is already a trivial clause)

Note that the fact that $p$ and $\neg p \vee q$ implies $q$ (called disjunctive syllogism) is a special case of resolution, since $p \vee F$ and $\neg p \vee q$ give us $F \vee q$ which is equivalent to $q$.

Resolution-based proof:

| Step | Reason |
|---|---|
| 1. $\neg s$ | hypothesis |
| 2. $\neg w \vee s$ | hypothesis |
| 3. $\neg w$ | resolution of 1 and 2 |
| 4. $w \vee t$ | hypothesis |
| 5. $t$ | resolution of 3 and 4 |
| 6. $\neg t \vee h$ | hypothesis |
| 7. $h$ | resolution of 5 and 6 |

# Fallacies

Fallacy = misconception resulting from incorrect argument.

- **Fallacy of affirming the conclusion**
  Based on
  $$((p \rightarrow q) \wedge q) \rightarrow p$$
  which is NOT A TAUTOLOGY.
  Ex.: If prof gives chocolate, then you answer the question. You answer the question. We conclude the prof gave chocolate.

- **Fallacy of denying the hypothesis**
  Based on
  $$((p \rightarrow q) \wedge \neg p) \rightarrow \neg q$$
  which is NOT A TAUTOLOGY.
  Ex.: If prof gives chocolate, then you answer the question. Prof doesn't give chocolate. Therefore, you don't answer the question.

# Rules of Inference for Quantified Statements

| Rule of Inference | Name |
|---|---|
| $\dfrac{\forall x P(x)}{\therefore\ P(c)}$ | Universal instantiation |
| $\dfrac{P(c) \text{ for an arbitrary } c}{\therefore\ \forall x P(x)}$ | Universal generalization |
| $\dfrac{\exists x P(x)}{\therefore\ P(c) \text{ for some element } c}$ | Existential instantiation |
| $\dfrac{P(c) \text{ for some element } c}{\therefore\ \exists x P(x)}$ | Existencial generalization |

Show that the premises:

- A student in Section A of the course has not read the book.
- Everyone in Section A of the course passed the first exam.

imply the conclusion

- Someone who passed the first exam has not read the book.

$A(x)$: "$x$ is in Section A of the course"
$B(x)$: "$x$ read the book"
$P(x)$: "$x$ passed the first exam."

Hypotheses: $\exists x(A(x) \wedge \neg B(x))$ and $\forall x(A(x) \rightarrow P(x))$.
Conclusion: $\exists x(P(x) \wedge \neg B(x))$.

Hypotheses: $\exists x(A(x) \wedge \neg B(x))$ and $\forall x(A(x) \rightarrow P(x))$.
Conclusion: $\exists x(P(x) \wedge \neg B(x))$.

| Step | Reason |
|---|---|
| 1. $\exists x(A(x) \wedge \neg B(x))$ | Hypothesis |
| 2. $A(a) \wedge \neg B(a)$ | Existencial instantiation from (1) |
| 3. $A(a)$ | Simplification from (2) |
| 4. $\forall x(A(x) \rightarrow P(x))$ | Hypothesis |
| 5. $A(a) \rightarrow P(a)$ | Universal instantiation from (4) |
| 6. $P(a)$ | Modus ponens from (3) and (5) |
| 7. $\neg B(a)$ | Simplification from (2) |
| 8. $P(a) \wedge \neg B(a)$ | Conjunction from (6) and (7) |
| 9. $\exists x(P(x) \wedge \neg B(x))$ | Existential generalization from (8) |

# Combining Rules of Inference for Propositions and Quantified Statements

These inference rules are frequently used and combine propositions and quantified statements:

- **Universal Modus Ponens**

$$\begin{array}{l} \forall x(P(x) \rightarrow Q(x)) \\ P(a), \text{ where } a \text{ is a particular element in the domain} \\ \hline \therefore \quad Q(a) \end{array}$$

- **Universal Modus Tollens**

$$\begin{array}{l} \forall x(P(x) \rightarrow Q(x)) \\ \neg Q(a), \text{ where } a \text{ is a particular element in the domain} \\ \hline \therefore \quad \neg P(a) \end{array}$$

# Proof Methods

A proof is a valid argument that establishes the truth of a mathematical statement, using the hypotheses of the theorem, if any, axioms assumed to be true, and previously proven theorems.

Using these ingredients and rules of inference, the proof establishes the truth of the statement being proved.

We move from formal proofs, as seen in the previous section, to **informal proofs**, where more than one inference rule may be used at each step, where steps may be skipped, and where axioms and rules of inference used are not explicitly stated.

# Some terminology

- **Theorem**: a statement that can be shown to be true (sometimes referred to as **facts** or **results**). Less important theorems are often called **propositions**.
- A **lemma** is a less important theorem, used as an auxiliary result to prove a more important theorem.
- A **corollary** is a theorem proven as an easy consequence of a theorem.
- A **conjecture** is a statement that is being proposed as a true statement. If later proven, it becomes a theorem, but it may be false.
- **Axiom** (or **postulates**) are statements that we assume to be true (algebraic axioms specify rules for arithmetic like commutative laws).
- A **proof** is a valid argument that establishes the truth of a theorem. The statements used in a proof include axioms, hypotheses (or premises), and previously proven theorems. Rules of inference, together with definition of terms, are used to draw conclusions from other assertions, tying together the steps of a proof.

# Understanding how theorems are stated

Many theorems assert that a property holds for all elements in a domain. However, the universal quantifier is often not explicitly stated.

The statement:

"If $x > y$, where $x$ and $y$ are positive real numbers, then $x^2 > y^2$."

really means

"For all positive real numbers $x$ and $y$, if $x > y$ then $x^2 > y^2$."

That is, in formal logic under the domain of positive real numbers this is the same as $\forall x \forall y((x > y) \rightarrow (x^2 > y^2))$.

# Methods of proving theorems

To prove a theorem of the form $\forall x(P(x) \rightarrow Q(x))$, we use the steps:

- Take an arbitrary element $c$ of the domain and show that $(P(c) \rightarrow Q(c))$ is true.
- Apply universal generalization to conclude $\forall x(P(x) \rightarrow Q(x))$. (Normally we not even bother with this final step.)

**3 methods of showing statements of the type $p \rightarrow q$ are true:**

1. **Direct proofs:** Assume $p$ is true; the last step establishes $q$ is true.
2. **Proof by Contraposition:** Uses a direct proof of the contrapositive of $p \rightarrow q$, which is $\neg q \rightarrow \neg p$. That is, assume $\neg q$ is true; the last step established $\neg p$ is true.
3. **Proof by Contradiction:** To prove that $P$ is true, we assume $\neg P$ is true and reach a contradiction, that is that $(r \wedge \neg r)$ is true for some proposition $r$. In particular, to prove $(p \rightarrow q)$, we assume $(p \rightarrow q)$ is false, and get as a consequence a contradiction. Assuming that $(p \rightarrow q)$ is false $= (\neg p \vee q)$ is false $= (p \wedge \neg q)$ is true.

# Direct Proofs

A **formal direct proof** of a conditional statement $p \rightarrow q$ works as follows: assume $p$ is true, build steps using inference rules, with the final step showing that $q$ is true.
In a **(informal) direct proof**, we assume that $p$ is true, and use axioms, definitions and previous theorems, together with rules of inference to show that $q$ must be true.

## Definition

The integer $n$ is *even* if there exists an integer $k$ such that $n = 2k$, and $n$ is *odd* if there exists an integer $k$ such that $n = 2k + 1$.

Give a direct proof of the following theorem.

## Theorem

*If $n$ is an odd integer, then $n^2$ is odd.*

**Theorem**

*If $n$ is an odd integer, then $n^2$ is odd.*

Observations: We want to show that $\forall n(P(n) \rightarrow Q(n))$, where $P(n)$ is "$n$ is an odd integer" and $Q(n)$ is "$n^2$ is odd".
We show this by proving that for an arbitrary $n$, $P(n)$ implies $Q(n)$, without invoking the universal generalization.

**Proof:**
Let $n$ be an odd integer.
By definition of odd, we know that there exists an integer $k$ such that $n = 2k + 1$.
Squaring both sides of the equation, we get
$n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$.
Since $n^2 = 2k' + 1$, where $k' = 2k^2 + 2k$, by the definition of odd we conclude $n^2$ is odd. $\square$

## Definition

An integer $a$ is a *perfect square* if there is an integer $b$ such that $a = b^2$.

**Exercise:**

Prove the following theorem using a direct proof.

## Theorem

*If $m$ and $n$ are both perfect squares, then $mn$ is also a perfect square.*

# Proof by Contraposition

This method of proof makes use of the equivalence $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$. In a proof by contraposition that $p \rightarrow q$, we assume $\neg q$ is true, and using axioms, definitions and previously proven theorems, together with inference rules, we show that $\neg p$ must be true. (It is a direct proof of the contrapositive statement!)

## Theorem

*If $n$ is an integer and $3n + 2$ is odd, then $n$ is odd.*

Proof: We prove the statement by contraposition.
Assume $n$ is even (assuming $\neg q$). Then, by definition, $n = 2k$ for some integer $k$. Thus, $3n + 2 = 3(2k) + 2 = 6k + 2 = 2(3k + 1)$. So, we have that $3n + 2 = 2k'$ where $k' = 3k + 1$, which means $3n + 2$ is an even number. This is the negation of the hypothesis of the theorem ($\neg p$), which concludes our proof by contraposition. $\square$

# Exercises: proof by contraposition

- Prove that if $n = ab$, where $a$ and $b$ are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.
- Show that the proposition $P(0)$ is true where the domain consists of the integer numbers and $P(n)$ is "If $n \geq 1$ then $n^2 > n$."
  Note: vacuous proof: when $p$ is false $p \rightarrow q$ is true, regardless of the value of $q$.

# When to use each type of proof?

Usually try a direct proof. If it doesn't work, try a proof by contraposition.

## Definition

A real number $r$ is *rational* if there exists integers $p$ and $q$ with $q \neq 0$ such that $r = p/q$. A real number that is not rational is called *irrational*

**Prove that the sum of two rational numbers is rational.** (For all $r, s \in \mathbb{R}$ , if $r$ and $s$ are rational numbers, then $r + s$ is rational.)
Let $r$ and $s$ be rational numbers. Then, there exist integers $p, q, t, u$ with $q \neq 0$ and $u \neq 0$ such that $r = p/q$ and $s = t/u$. So,

$$r + s = \frac{p}{q} + \frac{t}{u} = \frac{pu + qt}{qu}.$$

Since $q \neq 0$ and $u \neq 0$, we have $qu \neq 0$. Therefore, we expressed $r + s$ as the ratio of two integers $pu + qt$ and $qu$, where $qu \neq 0$. This means that $r + s$ is rational. (The direct proof succeeded!)

Prove that for any integer number $n$, if $n^2$ is odd, then $n$ is odd.

**Trying a direct proof...**
Let $n$ be an integer number. Assume that $n^2$ is odd. We get next that there exists an integer $k$ such that $n^2 = 2k + 1$. Solving for $n$ produces the equation $n = \pm\sqrt{2k+1}$, which is not very useful to show that $n$ is odd.

**Try a prove by contraposition...**
Let $n$ be an integer number. Assume $n$ is not odd. This means that $n$ is even, and so there exists an integer $k$ such that $n = 2k$. Thus, $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. So, taking $k' = 2k^2$, we see that $n^2 = 2k'$ and so $n^2$ is even. This concludes our proof by contraposition.

# Proof by Contradiction

In a proof by contradiction, we prove that a proposition $p$ is true, by showing that there exists a contradiction $q$ such that $\neg p \rightarrow q$.

We can prove that $p$ is true by showing for instance that $\neg p \rightarrow (r \wedge \neg r)$, for some proposition $r$.

**Prove that $\sqrt{2}$ is irrational.**

A direct proof is difficult, as it means to show that there exists no two integer $a$ and $b$ with $b \neq 0$ such that $\sqrt{2} = a/b$. Nonexistence proofs are usually hard.

Let's try a proof by contradiction...

## Theorem

*$\sqrt{2}$ is irrational.*

**Proof:** We prove by means of contradiction. Assume $\sqrt{2}$ is a rational number. So, there exists $a'$ and $b'$ integers with $b' \neq 0$ with $\sqrt{2} = a'/b'$. We select such integers $a$ and $b$ with the additional property that $a$ and $b$ have no common factors, i.e. the fraction $a/b$ is in lowest terms (this is always possible to obtain, for we can keep dividing by common factors). So, $\sqrt{2} = a/b$ so $2 = a^2/b^2$. This implies $2b^2 = a^2$ (1). By the definition of even, we know that $a^2$ is even. Next we use a theorem that states that if $a^2$ is even then $a$ is even (prove it as an exercise). Now, since $a$ is even, we know that there exists $c$ such that $a = 2c$. Substituting in the formula (1) above, we get that $2b^2 = (2c)^2 = 4c^2$. Dividing both sides by 2 we get $b^2 = 2c^2$. By the definition of even, we see that $b$ is even. Therefore, we got that $a$ is even and $b$ is even, and so 2 is a common factor of $a$ and $b$. But we also had that $a$ and $b$ had no common factors. We just reached a contradiction! $\square$

# Other types of proof statements

- **Proof of equivalences:**
  To prove a statement $p \leftrightarrow q$, we show that both $p \rightarrow q$ and $q \rightarrow p$ are true.

  Example: Prove that if $n$ is a positive integer, then $n$ is odd if and only if $n^2$ is odd.

- Showing that a statement of the form $\forall x P(x)$ is false:
  In this case, we need to find a **counterexample**.

  Example: Show that the statement "Every positive integer is the sum of the squares of two integers." is false.
  We argue that $3$ is a counterexample. The only perfect squares smaller than $3$ are $0$ and $1$, and clearly, $3$ cannot be written as a sum of two terms each being $0$ or $1$.

# Mistakes in Proofs

What is the problem with he following proof that $1 = 2$?

Use the following steps, where $a$ and $b$ are two equal positive integers.

| | |
|---|---|
| 1. $a = b$ | Given |
| 2. $a^2 = ab$ | Multiply both sides of (1) by $a$. |
| 3. $a^2 - b^2 = ab - b^2$ | Substract $b^2$ from both sides of (2) |
| 4. $(a - b)(a + b) = b(a - b)$ | Factoring both sides of (3) |
| 5. $a + b = b$ | Divide both sides of (4) by $a - b$ |
| 6. $2b = b$ | Replace $a$ by $b$ in (5), since $a = b$ |
| 7. $2 = 1$ | Divide both sides of 6. by $b$. |

Therefore $2 = 1$.

# Proof by Cases

Sometimes it is difficult to use a single argument that holds for all cases.
Proof by cases uses the following equivalence:

$$[(p_1 \vee p_2 \vee \cdots \vee p_n) \rightarrow q] \equiv [(p_1 \rightarrow q) \wedge (p_2 \rightarrow q) \wedge \cdots \wedge (p_n \rightarrow q)]$$

Example: Prove that if $n$ is integer then $n^2 \geq n$.
We split the proof into three cases.

- Case (i) $n = 0$.
  In this case, $n^2 = 0^2 = 0 = n$.
- Case (ii) $n \geq 1$
  In this case, when we multiply both sides of $n \geq 1$ by $n$ we obtain $n \cdot n \geq n \cdot 1$. This implies $n^2 \geq n$.
- Case (iii) $n \leq -1$
  In this case, $n \leq -1$, but $n^2 \geq 0$. Therefore, $n^2 \geq 0 \geq -1 \geq n$, and so $n^2 \geq n$.

□

# Exhaustive proof

This is a special form of a proof by cases, when there is a finite and small number of examples for which we need to prove a fact.

Prove that $(n+1)^2 \geq 3^n$ if $n$ is a positive integer with $n \leq 2$.

We use a proof by exhaustion, by examining the cases $n = 1, 2$.
For $n = 1$, $(n+1)^2 = 2^2 = 4 \geq 3 = 3^n$.
For $n = 2$, $(n+1)^2 = 3^2 = 9 \geq 3^2 = 3^n$.

# Existence Proofs

Existence proofs prove statements of the form $\exists x P(x)$.

- **Constructive existence proof: find $a$ such that $P(a)$ is true.**
  Example: Show that there is a positive integer that can be written as a sum of cubes of positive integers in two different ways.
  Proof: $1729 = 10^3 + 9^3$ and $1729 = 12^3 + 1^3$.

# Existence Proofs

Existence proofs prove statements of the form $\exists x P(x)$.

- **Constructive existence proof: find $a$ such that $P(a)$ is true.**
  Example: Show that there is a positive integer that can be written as a sum of cubes of positive integers in two different ways.
  Proof: $1729 = 10^3 + 9^3$ and $1729 = 12^3 + 1^3$.
- **Nonconstructive existence proof: show that $\exists x P(x)$ without explicitly giving $a$ for which $P(a)$ is true.**
  Example: Show that there exist irrational numbers $x$ and $y$ such that $x^y$ is rational.
  From a previous theorem we know that $\sqrt{2}$ is irrational. Consider the number $\sqrt{2}^{\sqrt{2}}$. There are two possible cases:

# Existence Proofs

Existence proofs prove statements of the form $\exists x P(x)$.

- **Constructive existence proof: find $a$ such that $P(a)$ is true.**
  Example: Show that there is a positive integer that can be written as a sum of cubes of positive integers in two different ways.
  Proof: $1729 = 10^3 + 9^3$ and $1729 = 12^3 + 1^3$.
- **Nonconstructive existence proof: show that $\exists x P(x)$ without explicitly giving $a$ for which $P(a)$ is true.**
  Example: Show that there exist irrational numbers $x$ and $y$ such that $x^y$ is rational.
  From a previous theorem we know that $\sqrt{2}$ is irrational. Consider the number $\sqrt{2}^{\sqrt{2}}$. There are two possible cases:
  - $\sqrt{2}^{\sqrt{2}}$ is rational: In this case, take $x = \sqrt{2}$ and $y = \sqrt{2}$.

# Existence Proofs

Existence proofs prove statements of the form $\exists x P(x)$.

- **Constructive existence proof: find $a$ such that $P(a)$ is true.**
  Example: Show that there is a positive integer that can be written as a sum of cubes of positive integers in two different ways.
  Proof: $1729 = 10^3 + 9^3$ and $1729 = 12^3 + 1^3$.
- **Nonconstructive existence proof: show that $\exists x P(x)$ without explicitly giving $a$ for which $P(a)$ is true.**
  Example: Show that there exist irrational numbers $x$ and $y$ such that $x^y$ is rational.
  From a previous theorem we know that $\sqrt{2}$ is irrational. Consider the number $\sqrt{2}^{\sqrt{2}}$. There are two possible cases:
  - $\sqrt{2}^{\sqrt{2}}$ is rational: In this case, take $x = \sqrt{2}$ and $y = \sqrt{2}$.
  - $\sqrt{2}^{\sqrt{2}}$ is irrational: In this case, take $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. Then $x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = (\sqrt{2})^2 = 2$.

# The art of finding a proof method that works for a theorem

We need practice in order to recognize which type of proof to apply to a particular theorem/fact that we need to prove.

In the next topic "number theory", several theorems will be proven using different proof methods and strategies.

# Introduction to Number Theory and its Applications

Lucia Moura

Winter 2010

*"Mathematics is the queen of sciences and the theory of numbers is the queen of mathematics." (Karl Friedrich Gauss)*

# Introduction

In the next sections we will review concepts from **Number Theory**, the branch of mathematics that deals with integer numbers and their properties.

We will be covering the following topics:

1. Divisibility and Modular Arithmetic (applications to hashing functions/tables and simple cryptographic cyphers). Section 3.4
2. Prime Numbers, Greatest Common Divisors (GCD) and Euclidean Algorithm. Section 3.5, part of 3.6
3. Applications to computer science: computer arithmetic with large integers and cryptography. Section 3.7

# Divisibility

When dividing an integer by a second nonzero integer, the quotient may or may not be an integer.
For example, $12/3 = 4$ while $9/4 = 2.25$.

The issue of divisibility is addressed in the following definition.

## Definition

If $a$ and $b$ are integers with $a \neq 0$, we say that $a$ *divides* $b$ if there exists an integer $c$ such that $b = ac$. When $a$ divides $b$ we say that $a$ is a *factor* of $b$ and that $b$ is a *multiple* of $a$.
The notation $a \mid b$ denotes $a$ divides $b$ and $a \nmid b$ denotes $a$ does not divide $b$.

Back to the above examples, we see that $3$ divides $12$, denoted as $3 \mid 12$, and $4$ does not divide $9$, denoted as $4 \nmid 9$.

# Divisibility Properties

## Theorem (1)

*Let $a, b$, and $c$ be integers. Then,*

1. *if $a \mid b$ and $a \mid c$ then $a \mid (b + c)$;*
2. *if $a \mid b$ then $a \mid bc$ for all integers $c$;*
3. *if $a \mid b$ and $b \mid c$ then $a \mid c$;*

Proof: Direct proof given in class.

## Corollary (1)

*If $a, b$, and $c$ are integers such that $a \mid b$ and $a \mid c$, then $a \mid mb + nc$ whenever $m$ and $n$ are integers.*

Proof: Direct proof given in class.

# The division algorithm

## Theorem (2, The division algorithm)

*Let $a$ be an integer and $d$ a positive integer. Then, there are unique integers $q$ and $r$, with $0 \leq r < d$, such that $a = dq + r$.*

- $d$ is called the *divisor*;
- $a$ is called the *dividend*;
- $q$ is called the *quotient*; this can be expressed $q = a \ \mathbf{div} \ d$;
- $r$ is called the *remainder*; this cane be expressed $r = a \ \mathbf{mod} \ d$;

**Example:**
If $a = 7$ and $d = 3$, then $q = 2$ and $r = 1$, since $7 = (2)(3) + 1$.
If $a = -7$ and $d = 3$, then $q = -3$ and $r = 2$, since $-7 = (-3)(3) + 2$.

# Using successive subtractions to find $q$ and $r$:

$a = 101$ and $d = 11$

| | |
|---|---|
| 101 | |
| - 11 | 46 |
| 90 | - 11 |
| - 11 | 35 |
| 79 | - 11 |
| - 11 | 24 |
| 68 | - 11 |
| - 11 | 13 |
| 57 | - 11 |
| - 11 | 2 |
| 46 | |

$q = 9$ as we subtracted $11$, $9$ times

$r = 2$ since this was the last value before getting negative.

# Proof of the previous theorem (the division Algorithm)

**Existence:**

Let $S$ be the set of nonnegative integers of the form $a - dq$, where $q$ is an integer. This set is nonempty because $-dq$ can be made as large as desired (taking $q$ as a negative integer with large absolute value). By the well-ordering property, $S$ has a least element $r = a - dq_0$ for some integer $q_0$.

The integer $r$ is nonnegative. It is also the case that $r < d$; otherwise if $r \geq d$, then there would be a smaller nonnegative element in $S$, namely $a - d(q_0 + 1)$, contradicting the fact that $a - dq_0$ was the smallest element of $S$. So, we just proved the existence of $r$ and $q$, with $0 \leq r < d$. $\square$

**Uniqueness:**

Suppose there exist $q, Q, r, R$ with $0 \leq r, R < d$ such that $a = dq + r$ and $a = dQ + R$. Assume without loss of generality that $q \leq Q$. Subtracting both equations, we have $d(q - Q) = (R - r)$. Thus, $d$ divides $(R - r)$, and so $|d| < |(R - r)|$ or $R - r = 0$. But we know that $0 \leq r, R < d$, so $|R - r| < d$, and we must have $R - r = 0$. This means $R = r$, which substituting into original equations gives $a - r = dq = dQ$. Since $d \neq 0$, dividing both sides of $dq = dQ$ by $d$ we get that $q = Q$. Therefore we have showed that $r = R$ and $q = Q$, proving uniqueness. $\square$

# Modular Arithmetic

## Definition

If $a$ and $b$ are integers and $m$ is a positive integer, then $a$ is *congruent to* $b$ *modulo* $m$ if $m$ divides $a - b$. We use the notation $a \equiv b \pmod{m}$ if this is the case, and $a \not\equiv b \pmod{m}$, otherwise.

The following theorem says that two numbers being congruent modulo $m$ is equivalent to their having the same remainders when dividing by $m$.

## Theorem (3)

*Let $a$ and $b$ be integers and let $m$ be a positive integer.*
*Then, $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$.*

Example: $10$ and $26$ are congruent modulo $8$, since their difference is $16$ or $-16$, which is divisible by $8$. When dividing $10$ and $26$ by $8$ we get $10 = 1 \cdot 8 + 2$ and $26 = 4 \cdot 8 + 2$. So $10 \bmod 8 = 2 = 16 \bmod 8$.

# Proof of the theorem given in class.

(you may use this space to take notes)

## Theorem (4)

*Let $m$ be a positive integer. The integers $a$ and $b$ are congruent modulo $m$ if and only if there is an integer $k$ such that $a = b + km$*

(Proof given in class.)

## Theorem (5)

*Let $m$ be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.*

(Proof given in class.)

## Corollary (2)

*Let $m$ be a positive integer and let $a$ and $b$ be integers. Then,*

$$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

$$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$$

Proof:
By the definition of $\bmod m$ and the definition of congruence modulo $m$, we know that $a \equiv (a \bmod m) \pmod m$, and $b \equiv (b \bmod m) \pmod m$.
Applying Theorem 5, we get

$$a + b \equiv (a \bmod m) + (b \bmod m) \pmod m$$

$$ab \equiv (a \bmod m)(b \bmod m) \pmod m$$

Using Theorem 3, from the above congruences we get the equalities in the statement of the theorem.

# Congruences and Hashing Functions

A **hashing table** is a data structure that allows for direct access to data. If done carefully, and under certain assumptions, we can search for a record with a set of $n$ records in expected time $O(1)$.
Each record is uniquely identified by a key (e.g. of keys are student number for a student record, account number for bank account records, call number for book records in a library, etc).

One of the most common hash functions uses modular arithmetic: $h(k) = k \bmod m$, where $m$ is the number of memory addresses.
Advantages: easy to compute, function is onto (all memory address can be used).

Since two different integers $k_1$ and $k_2$ may be mapped to the same location if $k_1 \equiv k_2 \pmod{m}$, **collisions** may arises. Methods for finding an alternate location for a key are employed (collision resolution techniques).

# Congruences and Pseudorandom Number Generators

We need random numbers in several types of algorithms, such as: **randomized algorithms**: algorithms that need to flip a coin to behave unbiasedly), **simulation algorithms**: where probability models are used to explain behaviour (example: arrival rate of subway passengers).
A systematic method of generating a number cannot be truly random, so we call them **pseudorandom number generators**. The most common method for such generators is the **linear congruential method**.
Pick integers $a$, $c$, $m$ and seed $x_0$, with $2 \leq a < m$, $0 \leq c, x_0 < m$.
Generate a sequence of numbers $x_0, x_1, x_2, \ldots$ from the seed $x_0$, using the congruence:

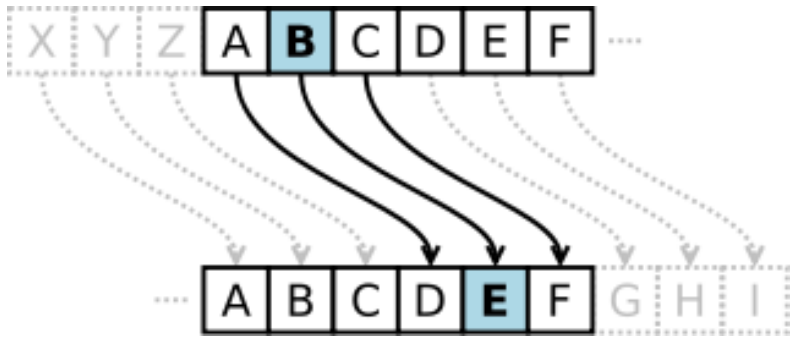
$$x_{n+1} = (ax_n + c) \bmod m.$$

The length of the period before repeats is called the **period**. Of course the period is at most $m$, and sometimes is exactly $m$ (see textbook example). For this reason $m$ must be large.
If we need a number in $[0, 1]$ we simply provide $x_n/m$.

# Congruences and Cryptography

Cryptology is the study of secret messages. One of its early uses was by Roman emperor Julius Caesar.

The Caesar cipher shifted each letter 3 letters forward in the alphabet (cyclically, sending xyz to abc respectively):




Decipher the message: JRRG OXFN LQ WKH PLGWHUP!

We can express the Caesar cipher mathematically using modular arithmetic (and generalizing the shift by $3$ to a shift by $k$):

encryption function: $f(p) = (p + k) \bmod 26$.

decryption function: $f^{-1}(p) = (p - k) \bmod 26$

# Primes

## Definition

A positive integer $p > 1$ is called *prime* if the only positive factors of $p$ are $1$ and $p$. A positive integer that is greater than one and is not prime is called *composite*.

An integer $n$ is composite if and only if there exists an integer $a$ such that $a|n$ and $1 < a < n$.

Prime numbers: $2$, $3$, $5$, $7$, $11$, $13$, $17$, etc.

For the following composite numbers $n$ provide a proof it is composite, that is, give a divisor $a$, with $1 < a < n$:
Composite Numbers: $4$, $6$, $8$, $9$, $10$, $12$, $14$, $15$, etc.

## Theorem (The Fundamental Theorem of Arithmetic)

*Every positive integer greater than* $1$ *can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size.*

The proof uses strong induction, so we will delay it until the next topic.

Examples:

$$\begin{aligned} 100 &= 2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2 \\ 641 &= 641 \\ 333 &= 3 \cdot 3 \cdot 37 = 3^2 \cdot 37 \\ 64 &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^6 \end{aligned}$$

## Theorem

*If $n$ is a composite integer, then $n$ has a prime divisor less than or equal to $\sqrt{n}$.*

Proof: If $n$ is composite then $n$ has a factor $a$ with $1 < a < n$. So, there exists an integer $b > 1$ such that $n = ab$. We claim that $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Indeed, assuming by contradiction that $a > \sqrt{n}$ and $b > \sqrt{n}$, we would get $n = ab > \sqrt{n} \cdot \sqrt{n} = n$, a contradiction. So, $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Thus, $n$ has a positive divisor $\leq \sqrt{n}$. If the divisor $d$ is prime, then the theorem follows. If the divisor $d$ is composite, then by the Fundamental Theorem of Arithmetic, it has a prime divisor $p < d \leq \sqrt{n}$, and since $p|d$ and $d|n$, we have that $p$ divides $n$, and the theorem follows in this case as well. $\square$

**Exercise: Use this Theorem to show $101$ is prime.**

**Theorem**

*There are infinitely many primes.*

Proof: We will use a proof by contradiction. Assume there are finitely many primes: $p_1, p_2, \ldots, p_n$. Let $Q = p_1 p_2 \cdots p_n + 1$.
By the Fundamental Theorem of Arithmetic, $Q$ is prime or it can be written as the product of two or more primes. In either case, there exists a prime $p$ such that $p|Q$.
We claim this prime $p$ cannot be any of the $p_i$ with $1 \leq i \leq n$. Indeed, if $p_i|Q$, we would conclude that $p_i|(Q - p_1 p_2 \cdots p_n) = 1$, which is a contradiction. Therefore, we conclude that $p$ is a prime, and is not any of the primes listed $p_1, p_2, \ldots, p_n$. But we have assumed that this was a complete list of all existing primes, and we reached a contradiction.
□

# Greatest Common Divisors

## Definition

Let $a$ and $b$ be integers, not both zero. The largest integer $d$ such that $d|a$ and $d|b$ is called the *greatest common divisor* of $a$ and $b$, and is denoted by $\gcd(a, b)$.

Example: The positive common divisors of $24$ and $36$ are $1, 2, 3, 4, 6, 12$. So, $\gcd(24, 36) = 12$.

Find the following greatest common divisors:
$\gcd(17, 100) =$
$\gcd(1000, 625) =$

## Definition

The integers $a$ and $b$ are *relatively prime* if $\gcd(a, b) = 1$.

$8$ and $9$ are relatively prime since $8 = 2^3$ and $9 = 3^2$, their only common divisor is $1$, giving $\gcd(8, 9) = 1$.

Are the following numbers relatively prime?

- $3$ and $12$
- $1024$ and $625$
- $7$ and $15$

## Proposition

*Let $a$ and $b$ be positive integers and let $p_1, p_2, \ldots, p_n$ be all the primes that appear in the prime factorization of $a$ or $b$, so that*

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \qquad b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n},$$

*where each $a_i, b_i \geq 0$ for $1 \leq i \leq n$. Then,*

$$\gcd(a,b) = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \cdots p_n^{\min(a_n,b_n)}$$

Proof: First note that the integer $d = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \cdots p_n^{\min(a_n,b_n)}$ divides $a$ and $b$, since the power of each prime $p_i$ does not exceed the power of $p_i$ appearing in the factorization of each of these numbers. Second, the exponents of $p_i$ in $d$ cannot be increased, otherwise it would not divide one of $a$ or $b$, and no other prime can be included. □

**Example:** $120 = 2^3 \cdot 3 \cdot 5$ and $500 = 2^2 \cdot 5^3$,
so $\gcd(120, 500) = 2^{\min(3,2)} 3^{\min(1,0)} 5^{\min(1,3)} = 2^2 3^0 5^1 = 20$.

## Definition

The *least common multiple* of the positive integers $a$ and $b$ is the smallest positive integer that is divisible by both $a$ and $b$, denoted by $\text{lcm}(a, b)$.

Examples: $\text{lcm}(2, 10) = 10$, $\text{lcm}(5, 7) = 35$, $\text{lcm}(4, 6) = 12$.

## Proposition

*Let $a$ and $b$ be positive integers and let $p_1, p_2, \ldots, p_n$ be all the primes that appear in the prime factorization of $a$ or $b$, so that*

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \qquad b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n},$$

*where each $a_i, b_i \geq 0$ for $1 \leq i \leq n$. Then,*

$$\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} p_2^{\max(a_2, b_2)} \cdots p_n^{\max(a_n, b_n)}$$

Proof: left as exercise (similar to the previous proposition)

## Theorem

*Let $a$ and $b$ be positive integers. Then,*

$$ab = \gcd(a,b) \cdot \operatorname{lcm}(a,b).$$

Proof: Write $a$ and $b$ as in the previous propositions

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \qquad b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n},$$

By these propositions, we have

$$\gcd(a,b) \cdot \operatorname{lcm}(a,b) =$$
$$(p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \cdots p_n^{\min(a_n,b_n)})(p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \cdots p_n^{\max(a_n,b_n)})$$
$$= p_1^{\max(a_1,b_1)+\min(a_1,b_1)} p_2^{\max(a_2,b_2)+\min(a_2,b_2)} \cdots p_n^{\max(a_n,b_n)+\min(a_n,b_n)}.$$

Now note that $\max(a_i, b_i) + \min(a_i, b_i) = a_i + b_i$. Thus,

$$\begin{aligned} \gcd(a,b) \cdot \operatorname{lcm}(a,b) &= p_1^{a_1+b_1} p_2^{a_2+b_2} \cdots p_n^{a_n+b_n} \\ &= p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n} p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n} = ab. \end{aligned}$$

# Towards an efficient GCD Algorithm

The methods described in Proposition 1 to calculate $\gcd(a, b)$ via the prime factorization of $a$ and $b$ is not efficient.
For instance, we do not know how to efficiently factor a number; that is, there are no polynomial time algorithm known that does this job. Since that method requires factoring $a$ and $b$, we would need to use algorithms that do not run in polynomial time (exponential or sub-exponential time).

Note that here the input size is $\lfloor \log_2 a \rfloor + \lfloor \log_2 b \rfloor$ (number of bits needed to represent $a$ and $b$). An algorithm running in linear time with $a$ and $b$ would not be a polynomial time algorithm.
However, there is an efficient algorithm which uses only $O(\log(\min(a, b)))$ integer divisions.
This algorithm was invented by Euclid, a famous mathematician living during 325-265 B.C.

# The Euclidean Algorithm

We want to calculate the gcd$(91, 287)$.
Applying the division algorithm to $287$ and $91$, we get

$$287 = 91 \cdot 3 + 14.$$

Any common divisor $d$ of $287$ and $91$ must also be a divisor of $14$, because $d|287$ and $d|91$ implies $d|(287 - 91 \cdot 3) = 14$.
Also, any common divisor of $91$ and $14$ must also be a divisor of $287$.
So, gcd$(287, 91)$=gcd$(91, 14)$. Great! We've just decreased one of the numbers. Continue the process by dividing $91$ by $14$:

$$91 = 14 \cdot 6 + 7.$$

Again, we conclude gcd$(91, 14)$=gcd$(14, 7)$, and divide $14$ by $7$:

$$14 = 7 \cdot 2 + 0$$

Because $7$ divides $14$ we know gcd$(14, 7) = 7$. Therefore,
gcd$(287, 91)$ =gcd$(91, 14)$ =gcd$(14, 7) = 7$.

The Euclidean algorithm is based on the following Lemma:

**Lemma**

*Let $a = bq + r$ where $a$, $b$, $q$ and $r$ are integers. Then gcd$(a, b)$ =gcd$(b, r)$.*

Proof: It is enough to show that the common divisors of $a$ and $b$ are the same as the common divisors of $b$ and $r$, for then they will also share the *greatest* common divisor.
Suppose $d|a$ and $d|b$. Then $d|a - bq = r$. So any common divisor of $a$ and $b$ is also a common divisor of $b$ and $r$.
Suppose $d|b$ and $d|r$. Then $d|bq + r = a$ So, any common divisor of $b$ and $r$ is a common divisor of $a$ and $b$.
Therefore, gcd$(a, b)$ =gcd$(b, r)$. □

# The Euclidean Algorithm

Correctness of the Algorithm: partial correctness +termination

```
Input:  a and b positive integers
Output:  gcd(a,b)
   x := max(a,b)
   y := min(a,b)
   gcd(x,y)=gcd(a,b)
   while (y≠0) do   Loop invariant:  gcd(x,y)=gcd(a,b)
     r := x mod y    by the previous THM: gcd(x,y)=gcd(y,r)
     x:=y
     y:=r
   endwhile (loop terminates since y≥ 0 and decreases at each iteration)
   postcondition gcd(x,y)=gcd(a,b) and y=0
   postcondition x=gcd(x,0)=gcd(a,b)
   return x
```

# Applying Euclidean Algorithm to find gcd(123, 277):

$$\begin{aligned} 277 &= 123 \cdot 2 + 31 \\ 123 &= 31 \cdot 3 + 30 \\ 31 &= 30 \cdot 1 + 1 \\ 30 &= \underline{1} \cdot 30 + 0 \\ \gcd(123, 277) &= 1 \end{aligned}$$

| x | 277 | 123 | 31 | 30 | **1** ← gcd |
|---|---|---|---|---|---|
| y | 123 | 31 =277 mod 123 | 30 =123 mod 31 | 1 =31 mod 30 | 0 =30 mod 1 |

# Useful Results

## Theorem (A)

*If $a$ and $b$ are positive integers, then there exist integers $s$ and $t$ such that $gcd(a, b) = sa + tb$.*

Example:
$\gcd(252, 198) = 18 = 4 \cdot 252 - 5 \cdot 198$

We won't prove this now, but will show a method for computing $s$ and $t$ called the extended Euclidean Algorithm.

# Extended Euclidean Algorithm

Consider the steps of the Euclidean algorithm for $\gcd(252, 198)$:

$$\begin{aligned} 252 &= 1 \cdot 198 + 54 \\ 198 &= 3 \cdot 54 + 36 \\ 54 &= 1 \cdot 36 + 18 \\ 36 &= 2 \cdot 18 \end{aligned}$$

Isolate the nonzero remainders in the above equations, substituting backwards:

$$\begin{aligned} \gcd(252, 198) = 18 &= 54 - 1 \cdot 36 \\ &= 54 - 1(198 - 3 \cdot 54) = 4 \cdot 54 - 1 \cdot 198 \\ &= 4 \cdot (252 - 1 \cdot 198) - 1 \cdot 198 = 4 \cdot 252 - 5 \cdot 198 \end{aligned}$$

Therefore, $\gcd(252, 198) = 4 \cdot 252 - 5 \cdot 198$.

## Lemma (A)

*If $a, b$, and $c$ are positive integers such that gcd$(a, b) = 1$ and $a|bc$, then $a|c$.*

Proof: Using Extended Euclidean Algorithm, there exists $s$ and $t$ such that $sa + tb = 1 =$gcd$(a, b)$. Multiplying by $c$, we get $sac + tbc = c$. Since $a|bc$ then $a|tbc$. But then by the above equation since $a|sac$ and $a|tbc$, we get that $a|c$. $\square$

The above Lemma generalizes to the following Lemma:

## Lemma (B)

*If $p$ is a prime and $p|a_1a_2 \cdots a_n$, where each $a_i$ is an integer, then $p|a_i$ for some $i$.*

Proof: Do as an exercise.

# Dividing both sides of a congruence

As we have seen, we can't always divide both sides of a congruence by the same integer, even if it is non-zero.

For example:
$6 \equiv 12 \pmod 6$, but $3 \not\equiv 6 \pmod 6$.
$14 \equiv 8 \pmod 6$, but $7 \not\equiv 4 \pmod 6$.

However, we can divide by appropriate integers $c$, as long as $\gcd(c, m) = 1$:

## Theorem (B)

*Let $m$ be a positive integer and let $a$, $b$, and $c$ be integers. If $ac \equiv bc \pmod m$ and $\gcd(c, m) = 1$, then $a \equiv b \pmod m$.*

Proof: Since $ac \equiv bc \pmod m$ we have that $m|ac - bc = c(a - b)$. By Lemma A, since $\gcd(c, m) = 1$, we have that $c|(a - b)$. This gives $a \equiv b \pmod m$. $\square$

# Solving Linear Congruences

Let $m$ be a positive integer, $a$ and $b$ be integers and $x$ be a variable. The following congruence is called a **linear congruence**:

$$ax \equiv b \pmod{m}.$$

How can we solve it, i.e. find all integers $x$ that satisfy it?

One possible method is to multiply both sides of the congruence by an inverse $\overline{a}$ of $a \pmod{m}$ if one such inverse exists:
$\overline{a}$ is an **inverse** of $a \pmod{m}$ if $\overline{a}a \equiv 1 \pmod{m}$.

Example:
$5$ is an inverse of $3 \pmod 7$, since $5 \cdot 3 \equiv 15 \equiv 1 \pmod 7$.
Using this we can solve:

$$\begin{aligned} 3x &\equiv 4 \pmod 7 \\ 5 \cdot 3x &\equiv 5 \cdot 4 \pmod 7 \\ 1 \cdot x &\equiv 20 \pmod 7 \\ x &\equiv 6 \pmod 7 \end{aligned}$$

Substitute back into the original linear congruence to check that $6$ is a solution:

$$3 \cdot 6 \equiv 18 \equiv 4 \pmod 7.$$

But how can we compute inverses $\pmod m$?

# Computing inverses modulo $m$

## Theorem

*If $a$ and $m$ are relatively prime integers with $m > 1$, then an inverse of $a$ modulo $m$ exists. Furthermore, this inverse is unique modulo $m$.*

Proof: By Theorem A, since $\gcd(a, m) = 1$, there exists $s$ and $t$ such that

$$sa + tm = 1.$$

This implies $sa + tm \equiv 1 \pmod{m}$. Since $tm \equiv 0 \pmod{m}$, so $sa \equiv 1 \pmod{m}$, which implies $s$ is an inverse of $a$ modulo $m$.
It remains to show that this inverse is unique modulo $m$. Suppose $s$ and $s'$ are inverses of $a$ modulo $m$. Then,

$$sa \equiv 1 \equiv s'a \pmod{m}.$$

Since $\gcd(a, m) = 1$, by Theorem B, we can divide both sides of the congruence by $a$, obtaining $s \equiv s' \pmod{m}$. $\square$

# Computing the inverse of 24 modulo 7

Applying the extended Euclidean Algorithm:

$$\begin{aligned} 24 &= 3 \cdot 7 + 3 \\ 7 &= 2 \cdot 3 + 1 \\ 3 &= 3 \cdot 1 + 0 \end{aligned}$$

Using backward substitution:

$$1 = 7 - 2 \cdot 3 = 7 - 2 \cdot (24 - 3 \cdot 7) = -2 \cdot 24 + 7 \cdot 7.$$

So $s = -2$ and $t = 7$.

$$-2 \cdot 24 \equiv 1 \pmod 7$$

You can use as an inverse of $24$ modulo $7$, any integer equivalent to $-2$ modulo $7$, such as: $\ldots, -9, -2, 5, 12, 19, \ldots$

# Chinese Remainder Thm: solving systems of congruences

A Chinese Mathematician asked in the first century:

> *There are certain things whose number is unknown. When divided by* $3$*, the remainder is* $2$*; when divided by* $5$ *the remainder is* $3$*; and when divided by* $7$*, the remainder is* $2$*. What will be the number of things?*

This puzzle is asking for the solution of the following system of congruences:

$$\begin{aligned} x &\equiv 2 \pmod 3, \\ x &\equiv 3 \pmod 5, \\ x &\equiv 2 \pmod 7. \end{aligned}$$

The Chinese Remainder Theorem establishes that when the moduli are pairwise relatively prime, we can solve such as system of linear congruences uniquely module the product of the moduli.

## Theorem (Chinese Reminder Theorem)

*Let $m_1, m_2, \ldots, m_n$ be pairwise relatively prime positive integers and $a_1, a_2, \ldots, a_n$ be arbitrary integers. Then, the system:*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ \ldots &\quad \ldots \\ x &\equiv a_n \pmod{m_n}, \end{aligned}$$

*has a unique solution modulo $m = m_1 m_2 \ldots m_n$. (That is, there is a solution $x$ with $0 \leq x < m$, and all other solutions are congruent modulo $m$ to this solution).*

# Proof of the Chinese Reminder Theorem (existence part)

In order to construct a simultaneous solution, let $M_k = m/m_k$. Note that $gcd(m_k, M_k) = 1$. So there exists $y_k$ inverse of $M_k$ modulo $m_k$.
Then $x = a_1M_1y_1 + a_2M_2y_2 + \cdots + a_nM_ny_n$ is a simulatneous solution.
Indeed, for any $1 \leq k \leq n$, since for $j \neq k$, all terms except $k$th term are $0$ modulo $m_k$, which gives $x \equiv a_2M_ky_k \equiv a_k \pmod{m_k}$. $\square$
*Showing that this is a unique solution is exercise 3.7-24, which is recommended.*
Solving the original old question, that asks for a simultaneous solution to $x \equiv 2 \pmod 3$, $x \equiv 3 \pmod 5$, $x \equiv 2 \pmod 7$.
$m_1 = 3$, $m_2 = 5$, $m_3 = 7$, so $m = m_1m_2m_3 = 105$.
$a_1 = 2$, $a_2 = 3$, $a_3 = 2$;
$M_1 = 35$, an inverse of $35$ modulo $3$: $2$ $M_2 = 21$, an inverse of $21$ modulo $5$: $1$ $M_3 = 15$. an inverse of $35$ modulo $3$: $1$
So the solution $x \equiv a_1M_1y_1 + a_2M_2y_2 + a_3M_3y_3 \equiv$
$2 \cdot 25 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 \equiv 233 \equiv 23 \pmod{105}$.

# Fermat's Little Theorem

## Theorem (Fermat's Little Theorem)

*If $p$ is a prime and $a$ is an integer not divisible by $p$, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

*Furthermore, for every integer $a$ we have*

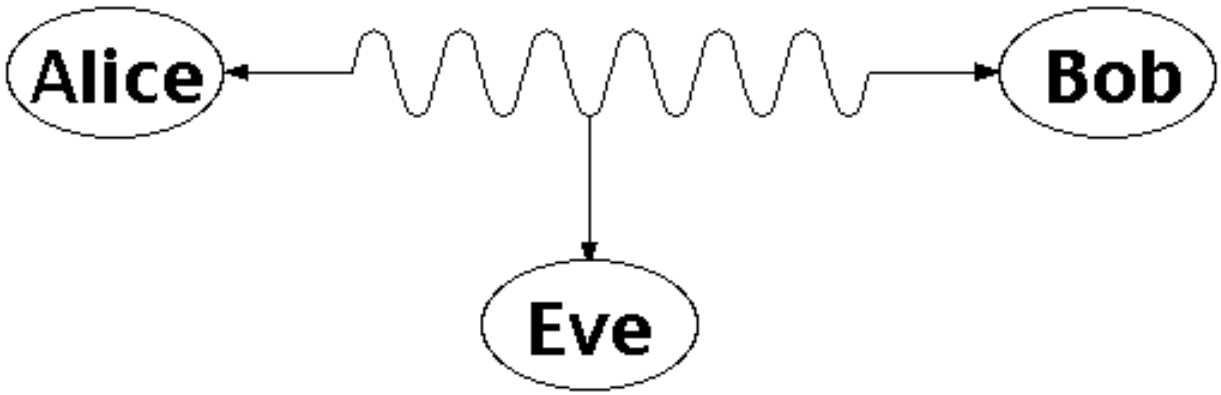
$$a^{p} \equiv a \pmod{p}.$$

The proof is left as an exercise, whose steps are outlined in Exercise 17 (page 244-245).

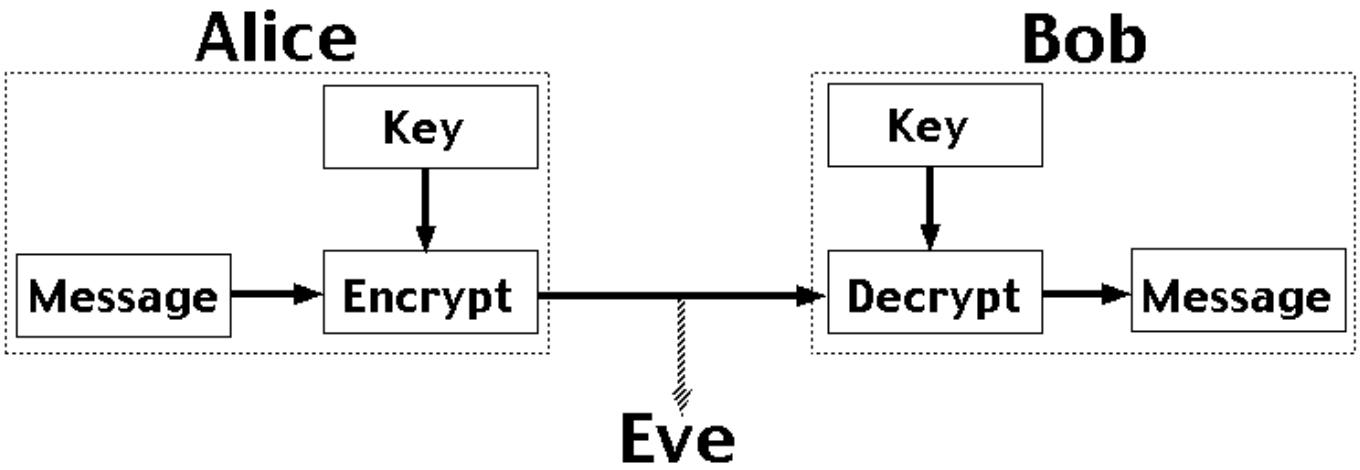
Example: $p = 5$
Verify that the theorem works for $a = 1, 2, 3, 4$: For 1 it is trivial, $2^4 = 16 \equiv 1 \pmod{5}$, $3^4 = 81 \equiv 1 \pmod{5}$, $4^4 = 256 \equiv 1 \pmod{5}$.

# Public Key Cryptography and the RSA Cryptosystem

Two people, say Alice and Bob, would like to exchange secret messages; however, Eve is eavesdropping:



One technique would be to use an encryption technique based on an **encryption key**, but this poses a challenge: how do they exchange the encryption key without Eve receiving it?



# Traditional Cryptography

In normal cryptography, both parties need to know a secret key $k$. The sender then encodes message $m$ using key $k$ via some method $f$ to get the ciphertext $c$:

$$c = f(m, k).$$

Then, the receiver decodes the ciphertext $c$ using key $k$ via some method $g$ to get back the original message $m$:

$$m = g(c, k).$$

The issue here is how to securely exchange the secret key $k$.

If we could find a method of encryption / decryption such that exchanging the necessary keys does not reveal to Eve how to decrypt intercepted messages, then we would avoid this problem altogether.

# Public Key Cryptosystem

The idea is that the receiver publically publishes a **public key** $k$. Then anyone who wishes to encode a message $m$ can do so:

$$c = f(m, k).$$

The receiver has a **private key** $k'$ that is needed to decode the ciphertext $c$ to receive the original message $m$:

$$m = g(c, k).$$

Both the encoding technique $f$ and the decoding technique $g$ are also publically known; the only secret information is $k'$.

While it is possible, it is computationally difficult to compute $k'$ from $k$: the **key pair** can be chosen so that in the amount of time it takes to derive $k'$ from $k$, the information $m$ no longer has significant value.

# RSA Cryptosystem

The most common form of public key cryptosystem is RSA, which stands for Rivest, Shamir, and Adleman, who invented it. It is based on modular arithmetic and large primes, and its security comes from the computational difficulty of factoring large numbers.

The idea is as follows: select $p$ and $q$ to be large primes (at least several hundred digits); the degree of security is dependent on the size of $p$ and $q$. Take $n = pq$. Then the **public key** is a pair $k = (n, e)$ such that:

$$\gcd(e, (p-1)(q-1)) = 1.$$

The **encoding function** is:

$$f(m, k) \equiv m^e \pmod{n}.$$

This assumes that the message can be represented by an integer $m < n$ with $\gcd(m, p) = \gcd(m, q) = 1$; if not, we can break $m$ down into smaller pieces and encode each individually.

The **private key** is a pair $k' = (n, d)$ such that:

$$de \equiv 1 \pmod{(p-1)(q-1)}.$$

The **decoding function** is:

$$g(c, k') = c^d \pmod{n}.$$

The security of the algorithm lies in the challenge of **prime factorization**: in order to calculate $d$, it is necessary to factor $n$ to get $p$ and $q$, which is very difficult (exponential in the number of digits in $p$ and $q$).

We now proceed to show that RSA actually works.

# Proof of the RSA Cryptosystem

## Theorem (RSA Cryptosystem)

*Let $p$, $q$ be primes with $n = pq$ and let $e$ be an integer such that $\gcd(e, (p-1)(q-1)) = 1$, with $ed \equiv 1 \pmod{(p-1)(q-1)}$. Let $m$ be an integer with $m < n$ and $\gcd(m, p) = \gcd(m, q) = 1$. Define $k = (n, e)$ and $k' = (n, d)$, and the functions:*

$$f(m, k) = m^d \pmod{n}$$

$$g(c, k) = c^e \pmod{n}.$$

*Then we claim that:*

$$g(f(m, k), k') = m.$$

## Proof.

We have that:

$$g(f(m,k),k') = (m^e \bmod n)^d \bmod n = m^{ed} \bmod n.$$

By the choice of $e$ and $d$, we have that: $ed \equiv 1 \pmod{(p-1)(q-1)}$, or, equivalently, for some integer $s$, $ed = 1 + s(p-1)(q-1)$. By Fermat's Little Theorem, $m^{p-1} \equiv 1 \pmod{p}$ and $m^{q-1} \equiv 1 \pmod{q}$, giving:

$$m^{ed} \equiv m^{1+s(p-1)(q-1)} \equiv m \cdot (m^{p-1})^{s(q-1)} \equiv m \cdot 1^{s(q-1)} \equiv m \pmod{p}.$$

Similarly, $m^{ed} \equiv m \pmod{q}$. Since $\gcd(p,q) = 1$, by the Chinese Remainder Theorem, $m^{ed} = m \pmod{pq}$ as required. □

Note that we can apply the same argument to show that:

$$f(g(m, k'), k) = m.$$

Thus, the owner of the private key can encrypt a message $m$ using the private key, which can then be decrypted by anyone using the public key, and prove that only the private key owner could have encrypted it. This is the basis of **digital signature systems**.

Example: Bob wants to receive messages from Alice, so he selects two primes, say $p = 43$ and $q = 59$. (We choose small primes for feasibility of the example; in reality, they would be vastly larger.) Then $n = pq = 2537$ and $(p-1)(q-1) = 2436$. He then picks $e = 13$, which has the property that:

$$\gcd(e, (p-1)(q-1)) = \gcd(13, 2436) = 1.$$

Bob then calculates $d = 937$, the inverse of $e$ mod $2436$:

$$de \equiv 937 \times 13 \equiv 12181 \equiv 5 \times 2436 + 1 \equiv 1 \pmod{2436}.$$

Bob publishes the **public key** $k = (2537, 13)$.

Alice wants to send message "STOP" to Bob using RSA. She encodes this: S $\rightarrow 18$, T $\rightarrow 19$, O $\rightarrow 14$, P $\rightarrow 15$, i.e. 1819 1415 grouped into blocks of 4. Thus, $m = m_1 m_2 = 18191415$. Each block is encrypted:

$$1819^{13} \text{ mod } 2537 = 2081$$

$$1451^{13} \text{ mod } 2537 = 2182$$

Then the encrypted message is $20812182$. Bob has **private key** $k' = (2537, 937)$, and computes:

$$2081^{937} \bmod 2537 = 1819 \rightarrow \text{ST}$$

$$2812^{937} \bmod 2537 = 1415 \rightarrow \text{OP}$$

Thus, the original message was STOP.

# Induction and Recursion

Lucia Moura

Winter 2010

# Mathematical Induction

## Principle (of Mathematical Induction)

*Suppose you want to prove that a statement about an integer $n$ is true for every positive integer $n$.*

- *Define a propositional function $P(n)$ that describes the statement to be proven about $n$.*
- *To prove that $P(n)$ is true for all $n \geq 1$, do the following two steps:*
  - *Basis Step: Prove that $P(1)$ is true.*
  - *Inductive Step: Let $k \geq 1$. Assume $P(k)$ is true, and prove that $P(k+1)$ is true.*

# Types of statements that can be proven by induction

1. **Summation formulas**
   Prove that $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$, for all integers $n \geq 0$.
2. **Inequalities**
   Prove that $2^n < n!$ for every positive integer $n$ with $n \geq 4$.
3. **Divisibility results**
   Prove that $n^3 - n$ is divisible by $3$ for every positive integer $n$.
4. **Results about sets**
   Prove that if $S$ is a set with $n$ elements where $n$ is a nonnegative integer, then $S$ has $2^n$ subsets.
5. **Creative use of mathematical induction**
   Show that for $n$ a positive integer, every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes (L shape).
6. **Results about algorithms**
   Prove that procedure $\text{FAC}(n)$ returns $n!$ for all nonnegative integers $n \geq 0$.

Prove that algorithm $\text{FAC}(n)$ returns $n!$ for all nonnegative integers $n \geq 0$.

**procedure** $\text{FAC}(n$: nonnegative integer)
    **if** $(n = 0)$ **then return** $1$
        **else return** $n * \text{FAC}(n-1)$

# Strong Induction

## Principle (of Strong Induction)

*Suppose you want to prove that a statement about an integer $n$ is true for every positive integer $n$.*

- *Define a propositional function $P(n)$ that describes the statement to be proven about $n$.*
- *To prove that $P(n)$ is true for all $n \geq 1$, do the following two steps:*
  - *Basis Step: Prove that $P(1)$ is true.*
  - *Inductive Step: Let $k \geq 1$. Assume $P(1), P(2), \ldots, P(k)$ are all true, and prove that $P(k+1)$ is true.*

Use strong induction to prove:

**Theorem (The Fundamental Theorem of Arithmetic)**

*Every positive integer greater than* $1$ *can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size.*

**Proof:**

**Part 1:** Every positive integer greater than $1$ can be written as a prime or as the product of two or more primes.

**Part 2:** Show uniqueness, when the primes are written in nondecreasing order.

**Proof of Part 1:** Consider $P(n)$ the statement "$n$ can be written as a prime or as the product of two or more primes.". We will use strong induction to show that $P(n)$ is true for every integer $n \geq 1$.

BASIS STEP: $P(2)$ is true, since 2 can be written as a prime, itself.

INDUCTION STEP: Let $k \geq 2$. Assume $P(1), P(2), \ldots, P(k)$ are true. We will prove that $P(k+1)$ is true, i.e. that $k+1$ can be written as a prime or the product of two or more primes.

Case 1: $k+1$ is prime.

If $k+1$ is prime, then the statement is true as $k+1$ can be written as itself, a prime.

Case 2: $k+1$ is composite.

By definition, there exist two positive integers $a$ and $b$ with $2 \leq a \leq b < k+1$, such that $k+1 = ab$. Since $a, b < k+1$, we know by induction hypothesis that $a$ and $b$ can each be written as a prime or the product of two or more primes. Thus, $k+1 = ab$ can be written as a product of two or more primes, namely those primes in the prime factorization of $a$ and those in the prime factorization of $b$.

We want to prove Part 2. The following Lemma has been proven.

## Lemma (A)

*If $a, b$, and $c$ are positive integers such that gcd$(a, b) = 1$ and $a|bc$, then $a|c$.*

We prove the following lemma using induction.

## Lemma (B)

*If $p$ is a prime and $p|a_1a_2 \cdots a_n$, where each $a_i$ is an integer and $n \geq 1$, then $p|a_i$ for some $i$, $1 \leq i \leq n$.*

**Proof:** Let $P(n)$ be the statement "If $a_1, a_2, \ldots, a_n$ are integers and $p$ is a prime number such that $p|a_1a_2 \cdots a_n$, then $p|a_i$ for some $i$, $1 \leq i \leq n$". We will prove $P(n)$ is true for all $n \geq 1$.

Let $P(n)$ be the statement "If $a_1, a_2, \ldots, a_n$ are integers and $p$ is a prime number such that $p|a_1 a_2 \cdots a_n$, then $p|a_i$ for some $i$, $1 \leq i \leq n$".
We will prove $P(n)$ is true for all $n \geq 1$.

Basis: Prove that $P(1)$ is true.
The statement is trivially true, since for $n = 1$, $p|a_1$ already gives that $p|a_i$ for some $i$, $1 \leq i \leq 1$.

Induction step: Let $n \geq 2$. Assume $P(n-1)$ is true. Prove $P(n)$ is true.
Let $p$ be a prime such that $p|a_1 a_2 \cdots a_n$. In the case that $p|a_n$, we are done. So, consider the case $p \not| a_n$. Since $p$ is prime, we have $\gcd(p, a_n) = 1$, thus, by Lemma A, $p|a_1 \ldots a_{n-1}$. By induction hypothesis, we have that $p|a_i$ for some $i$, $1 \leq i \leq n-1$. Combining both cases, we get that $p|a_i$ for some $i$, $1 \leq i \leq n$.
□

# Proof of Part 2: (uniqueness of the prime factorization of a positive integer).

Suppose by contradiction that $n$ can be written as a product of primes in two different ways, say $n = p_1 p_2 \dots p_s$ and $n = q_1 q_2 \dots q_t$, where each $p_i$ and $q_j$ are primes such that $p_1 \leq p_2 \leq \cdots \leq p_s$ and $q_1 \leq q_2 \leq \cdots \leq q_t$.
When we remove all common primes from the two factorizations, we have: $p_{i_1} p_{i_2} \cdots p_{i_u} = q_{j_1} q_{j_2} \cdots q_{j_v}$, where no primes occur on both sides of this equations and $u$ and $v$ are positive integers.
By Lemma B, $p_{i_1}$ must divide $q_{j_k}$ for some $k$, $1 \leq k \leq v$. Since $p_{i_1}$ and $q_{j_k}$ are primes we must have $p_{i_1} = q_{j_k}$, which contradicts the fact that no primes appear on both sides of the given equation.
□

# Examples of statements that can be proven by strong induction

1. Consider a game with 2 players that take turns removing any positive number of matches they want from one of two piles of matches. The player that removes the last match wins the game. Prove that if both piles contains the same number of matches initially, the second player can always guarantee a win.
2. Prove that every amount of 12 cents or more can be formed with 4-cent and 5-cent stamps. (also try to prove it in a different way using mathematical induction)
3. Prove that algorithm $\text{GCD}(a, b)$ (given in page 313 of the textbook) returns the gcd$(a, b)$ for all integers $a, b$, $a < b$.

Prove that procedure $\text{GCD}(a, b)$ returns the gcd$(a, b)$ for all integers $a, b$, $a < b$.

**procedure** $\text{GCD}(a, b$: nonnegative integers with $a < b)$
    **if** $(a = 0)$ **then return** $b$
        **else return** $\text{GCD}(b \bmod a, a)$

# Recursive Definitions

We can use recursion to define:

- functions,
- sequences,
- sets.

Mathematical induction and strong induction can be used to prove results about recursively defined sequences and functions.

Structural induction is used to prove results about recursively defined sets.

# Recursively Defined Functions

Examples:

- Defining the factorial function recursively:

$$\begin{aligned} F(0) &= 1, \\ F(n) &= n \times F(n-1), \text{ for } n \geq 1. \end{aligned}$$

- Defining the maximum number of comparisons for the Mergesort algorithm (given in page 318):

$$\begin{aligned} T(1) &= 0, \\ T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1, \text{ for } n \geq 2. \end{aligned}$$

- Number of moves needed to solve the Hanoi tower problem:

$$\begin{aligned} H(1) &= 1, \\ H(n) &= 2H(n-1) + 1, \text{ for } n \geq 2. \end{aligned}$$

# Recursively Defined Sequences

Consider the Fibonacci numbers, recursively defined by:

$$\begin{aligned} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2}, \text{ for } n \geq 2. \end{aligned}$$

Prove that whenever $n \geq 3$, $f_n > \alpha^{n-2}$ where $\alpha = (1 + \sqrt{5})/2$.

Let $P(n)$ be the statement "$f_n > \alpha^{n-2}$". We will show that $P(n)$ is true for $n \geq 3$ using strong induction.

BASIS: We show that $P(3)$ and $P(4)$ are true:
$\alpha = (1+\sqrt{5})/2 < (1+3)/2 = 2 = f_3$.
$\alpha^2 = ((1+\sqrt{5})/2)^2 = (1^2+2\sqrt{5}+5)/4 = (3+\sqrt{5})/2 < (3+3)/2 = 3 = f_4$.

INDUCTIVE STEP: Let $k \geq 4$. Assume $P(j)$ is true for all integers $j$ with $3 \leq j \leq k$. Prove that $P(k+1)$ is true.
We have:

$$\begin{aligned} f_{k+1} &= f_k + f_{k-1}, \quad \text{(by the definition of the Fibonacci sequence)} \\ &> \alpha^{k-2} + \alpha^{k-3}, \quad \text{(by induction hypothesis)} \\ &= \alpha^{k-3}(\alpha+1) = \alpha^{k-3}((1+\sqrt{5})/2+1) = \alpha^{k-3}((3+\sqrt{5})/2) \\ &= \alpha^{k-3}\alpha^2 = \alpha^{k-1}. \end{aligned}$$

# Recursively Defined Sets and Structures

## Definition (Set of strings over an alphabet)

The set $\Sigma^*$ of strings over the alphabet $\Sigma$ can be defined recursively by:
BASIS STEP: $\lambda \in \Sigma^*$ (where $\lambda$ is the empty string)
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Example: If $\Sigma = \{0, 1\}$, then
$\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$.

## Definition (Well-formed formulas of Operators and Operands)

BASIS STEP: $x$ is a well-formed formula if $x$ is a numeral or variable.
RECURSIVE STEP: If $F$ and $G$ are well-formed formulas, then $(F + G)$, $(F - G)$, $(F * G)$, $(F/G)$ and $(F \uparrow G)$ are well-formed formulas.

Example: The following are well-formed formulas:
$(x * 3)$, $(3/0)$, $((x + 2) * y)$, $((2 + 3) - (x/y))$, etc.

# Structural Induction

Structural induction is used to show results about recursively defined sets.

## Principle (of Structural Induction)

*To show that a statement holds for all elements of a recursively defined set, use the following steps:*

- BASIS STEP: *Prove that the statement holds for all elements specified in the basis step of the set definition.*
- RECURSIVE STEP: *Prove that if the statement is true for each of the elements used to construct elements in the recursive step of the set definition, then the result holds for these new elements.*

The validity of this principle comes from the validity of mathematical induction, as we can transform the above argument on an induction on $n$ where $n$ is the number of applications of the recursive step of the set definition needed to obtain the element we are analysing.

# Example of Structural Induction I

Prove that every well-formed formula of Operators and Operands contains an equal number of left and right parentheses.

**Proof by structural induction:**

BASIS STEP: A numeral or a variable, each contains 0 parentheses, so clearly they contain an equal number of right and left parentheses.

RECURSIVE STEP: Assume $F$ and $G$ are well-formed formulas each containing an equal number of left and right parentheses. That is, if $l_F$ and $l_G$ are the number of left parentheses in $F$ and $G$, respectively, and $r_F$ and $r_G$ are the number of right parentheses in $F$ and $G$, respectively, then $l_F = r_F$ and $l_G = r_G$. We need to show that $(F + G)$, $(F - G)$, $(F * G)$, $(F/G)$ and $(F \uparrow G)$ also contain an equal number of left and right parenthesis. For each of these well-formed formulas, the number of left parentheses is $L = l_F + l_G + 1$ and the number of right parentheses is $R = r_F + r_G + 1$. Since $l_F = r_F$ and $l_G = r_G$, it follows that $L = l_F + l_G + 1 = r_F + r_G + 1 = R$. This concludes the inductive proof. □

# Example of Structural Induction II

Recall the definition of a set of strings.

## Definition (Set of strings over an alphabet)

The set $\Sigma^*$ of strings over the alphabet $\Sigma$ can be defined recursively by:
BASIS STEP: $\lambda \in \Sigma^*$ (where $\lambda$ is the empty string)
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

We now give a definition of concatenation of two strings.
Note how this definition is built on the definition of string.

## Definition (Concatenation of two strings)

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.
RECURSIVE STEP: If $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$ and $x \in \Sigma$, then
$w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

We now give a recursive definition of the reversal of a string.

## Definition (Reversal of a string)

BASIS STEP: $\lambda^R = \lambda$
RECURSIVE STEP: If $w \in \Sigma^*$ and $x \in \Sigma$, then $(wx)^R = x \cdot (w)^R$.

**Exercise:** Use structural induction to prove that if $w_1$ and $w_2$ are strings, then $(w_1 \cdot w_2)^R = w_2^R \cdot w_1^R$.

Note that this proof needs to use the 3 definitions given above.

# Proving the correctness of recursive programs

Mathematical induction (and strong induction) can be used to prove that a recursive algorithm is correct:

to prove that the algorithm produces the desired output for all possible input values.

We will see some examples next.

# Recursive algorithm for computing $a^n$

**procedure** power($a$: nonzero real number, $n$: nonnegative integer)
  **if** $(n = 0)$ **then** return $1$
    **else** return $a \times$power$(a, n - 1)$

We will prove by mathematical induction on $n$ that the algorithm above is correct.
We will show $P(n)$ is true for all $n \geq 0$, for
$P(n)$: For all nonzero real numbers $a$, power$(a, n)$ correctly computes $a^n$.

# Proving $power(a, n)$ is correct

**Basis:** If $n = 0$, the first step of the algorithm tells us that power(a,0)=1. This is correct because $a^0 = 1$ for every nonzero real number $a$, so $P(0)$ is true.

**Inductive step:**
Let $k \geq 0$.
Inductive hypothesis: power$(a, k) = a^k$ , for all $a \neq 0$.
We must show next that power$(a, k + 1) = a^{k+1}$.
Since $k + 1 > 0$ the algorithm sets power$(a, k + 1) = a \times$power$(a, k)$.
By inductive hypotheses $power(a, k) = a^k$, so
power$(a, k + 1) = a \times$power$(a, k) = a \times a^k = a^{k+1}$.

# Recursive algorithm for computing $b^n \mod m$

procedure mpower($b, n, m$: integers with $m \geq 2, n \geq 0$)
  if $n = 0$ then return 1;
  else if $n$ is even then return mpower$(b, n/2, m)^2 \mod m$
  else return ((mpower$(b, \lfloor n/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$

Examples:

$$
\begin{aligned}
& power(2,5,6) = \\
= \; & ((power(2,2,6)^2 \mod 6) * (2 \mod 6)) \mod 6 \\
= \; & (((power(2,1,6)^2 \mod 6)^2 \mod 6) * (2)) \mod 6 \\
= \; & ((((power(2,0,6)^2 \mod 6) * (2 \mod 6)) \mod 6)^2 \mod 6)^2 \mod 6 \\
& * 2) \mod 6 \\
= \; & ((((1^2 \mod 6) * 2) \mod 6)^2 \mod 6)^2 \mod 6) * 2) \mod 6 \\
= \; & 2
\end{aligned}
$$

# Proving $mpower(a, n, m)$ is correct, using induction on $n$

**Basis:** Let $b$ and $m$ be integers with $m \geq 2$, and $n = 0$. In this case, the algorithm returns 1. This is correct because $b^0 \mod m = 1$.

**Inductive step:**

Induction hypothesis: Let $k \geq 1$. Assume power$(b, j, m) = b^j \mod m$ for all integers $j$ with $0 \leq j \leq k-1$, whenever $b$ is a positive integer and $m$ is an integer with $m \geq 2$.

We must show next that power$(b, k, m) = b^k \mod m$. There are two cases to consider.

- Case 1: $k$ is even. In this case, the algorithm returns mpower$(b, k/2, m)^2 \mod m = (i.h.)(b^{k/2} \mod m)^2 \mod m = b^k \mod m$.
- Case 2: $k$ is odd. In this case, the algorithm returns $((\text{mpower}(b, \lfloor k/2 \rfloor, m)^2 \mod m) * (b \mod m)) \mod m$
  $= (i.h.)(b^{\lfloor k/2 \rfloor} \mod m)^2 \mod m) * (b \mod m)) \mod m$
  $= (b^{2\lfloor k/2 \rfloor + 1} \mod m) = b^k \mod m$.

# Program verification

We want to be able to prove that a given program meets the intended specifications.
This can often be done manually, or even by automated program verification tools. One example is PVS (People's Verification System).

A program is **correct** if it produces the correct output for every possible input.
A program has **partial correctness** if it produces the correct output for every input for which the program eventually halts.
Therefore, a program is correct if and only if it has partial correctnes and terminates.

# Hoare's triple notation

- A program's I/O specification can be given using initial and final assertions.
  - The **initial assertion** $p$ is the condition that the program's input (its initial state) is guaranteed (by its user) to satisfy.
  - The **final assertion** $q$ is the condition that the output produced by the program (its final state) is required to satisfy.
- Hoare triple notation:
  - The notation $p\{S\}q$ means that, for all inputs $I$ such that $p(I)$ is true, if program $S$ (given input $I$) halts and produces output $O = S(I)$, then $q(O)$ is true.
  - That is, $S$ is partially correct with respect to specification $p, q$.

# A simple example

- Let $S$ be the program fragment
  `y:= 2; z:= x+y`
- Let $p$ be the initial assertion `x=1`.
  The variable `x` will hold 1 in all initial states.
- Let $q$ be the final assertion `z= 3`.
  The variable `z` must hold 3 in all final states.
- Prove $p\{S\}q$.
  Proof: If `x`=1 in the program's input state, then after running `y`:=2 and `z`:=`x`+`y`, then `z` will be $1 + 2 = 3$.

# Rules of inference for Hoare triples

- The composition rule:

$$\frac{p\{S_1\}q \quad q\{S_2\}r}{\therefore p\{S_1; S_2\}r}$$

- It says: If program $S_1$ given condition $p$ produces condition $q$, and $S_2$ given $q$ produces $r$,
  then the program "$S_1$ followed by $S_2$", if given $p$, yields $r$.

# Inference rule for **if-then** statements

$$\frac{(p \wedge cond)\{S\}q \quad (p \wedge \neg cond) \rightarrow q}{\therefore p\{\textbf{if } cond \textbf{ then } S\}q}$$

Example: Show that: $T\{\textbf{if } x > y \textbf{ then } y := x\}(y \geq x)$.

Proof:
When initially $T$ is true, if $x > y$, then the **if**-body is executed, setting $y = x$, and so afterwards $y \geq x$ is true. Otherwise, $x \leq y$ and so $y \geq x$. In either case, the final assertion $y \geq x$ is true. So the rule applies, and so the fragment meets the specification.

# Inference rule for **if-then-else** statements

$$\frac{(p \wedge cond)\{S_1\}q \quad (p \wedge \neg cond)\{S_2\}q}{\therefore p\{\textbf{if } cond \textbf{ then } S_1 \textbf{ else } S_2\}q}$$

Example: Prove that
$T\{\textbf{if } x < 0 \textbf{ then } abs := -x \textbf{ else } abs := x\}(abs = |x|)$

Proof:
If the initial assertion is true and $x < 0$ then after the **if**-body, $abs$ will be $-x = |x|$.
If the initial assertion is true, but $\neg(x < 0)$ is true, i.e., $x \geq 0$, then after the **else**-body, $abs = x$, which is $|x|$.
So using the above rule, we get that this segment is true with respect to the final assertion.

# Loop Invariants

For a **while**-loop "**while** cond $S$", we say that $p$ is a **loop invariant** of this loop if $(p \wedge cond)\{S\}p$.

If $p$ (and the continuation condition $cond$) is true before executing the body, then $p$ remains true afterwards.

And so $p$ stays true through all subsequent iterations.

This leads to the inference rule:

$$\frac{(p \wedge cond)\{S\}p}{\therefore p\{\textbf{while } cond\ S\}(\neg cond \wedge p)}$$

# Example1: loop invariant

Prove that the following Hoare triple holds:
$T\{i := 1; fact := 1; \textbf{while } i < n\{i++; fact = fact * i\}\}(fact = n!)$

Proof:
Let $p$ be the assertion "'$fact = i! \wedge i \leq n$". We will show tht $p$ is a loop invariant.
Assume that at the beginning of the **while**-loop $p$ is true and the condition of the **while**-loop holds, in other words, assume that $fact = i!$ and $i < n$.
The new values $i_{new}$ and $fact_{new}$ of $i$ and $fact$ are
$i_{new} = i + 1$ and
$fact_{new} = fact \times (i + 1) = (i!) \times (i + 1) = (i + 1)! = i_{new}!$.
Since $i < n$, we also have $i_{new} = i + 1 \leq n$.
Thus $p$ is true at the end of the execution of the loop. This shows $p$ is a loop invariant.

# Final example: combining all rules

procedure multiply$(m, n : integers)$

$p :=$ "$(m, n \in Z)$"

**if** $n < 0$ **then** $a := -n$ (segment $S_1$)

**else** $a := n$

$q :=$ "$(p \wedge (a = |n|))$"

$k := 0; x := 0$ (segment $S_2$)

$r :=$ "$q \wedge (k = 0) \wedge (x = 0)$"

$(x = mk \wedge k \leq a)$

**while** $k < a$ { (segment $S_3$)

$x = x + m; \quad k = k + 1;$

}Maintains loop invariant: $(x = mk \wedge k \leq a)$

$(x = mk \wedge k = a) \therefore s :=$ "$(x = ma) \wedge a = |n|$"

$s \Rightarrow (n < 0 \wedge x = -mn) \vee (n \leq 0 \wedge x = mn)$

**if** $n < 0$ **then** $prod := -x$ (segment $S_4$)

**else** $prod := x$

$t =$ "$(prod = mn)$"

# Correctness of multiply$(m, n)$

The proof is structured as follows, by using propositions $p, q, r, s, t$ as defined in the previous page.

- Prove $p\{S_1\}q$ by using **if-then-else** inference rule.
- Prove $q\{S_2\}r$ by examining this trivial segment.
- Prove $r\{S_3\}s$ by using **while-loop** inference rule.
- Prove $s\{S_4\}t$ by using **if-then-else** inference rule.
- Use the rule of composition to show that $p\{S_1; S_2; S_3; S_4\}t$; recall that $p :=$ "$(m, n \in Z)$" and $t =$ "$(prod = mn)$", which is what we wanted to show for the partial correctness.

To complete the proof of correctness, given the partial correctness, we must verify that each segment terminates.
Termination is trivial for segments $S_1$, $S_2$ and $S_4$; for the **while-loop** $(S_4)$ it is easy to see that it runs for $a$ iterations.
(See general rule for proving termination of loops in the next page)
We leave the details of each step above as an exercise.

# Proving termination of a loop

- Associate with each iteration $i$ a natural number $k_i$, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.

# Proving termination of a loop

- Associate with each iteration $i$ a natural number $k_i$, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.

# Proving termination of a loop

- Associate with each iteration $i$ a natural number $k_i$, such that $< k_0, k_1, k_2, \ldots >$ is a decreasing sequence.
- Using the well-ordering principle, every decreasing sequence of natural numbers is finite.
- Find a decreasing sequence of natural numbers for the while-loop in the previous example:

  Define $k_i = a - k$
  $< k_0, k_1, k_2, \ldots >$ is decreasing as $a$ is constant and $k$ increases by $1$ at each iteration.

# Recurrence Relations

Lucia Moura

Winter 2010

# Recurrence Relations

- 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 $a_0, a_1, \ldots, a_{n-1}$, for all integers $n$ with $n \geq n_0$.

# Recurrence Relations

- 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 $a_0, a_1, \ldots, a_{n-1}$, for all integers $n$ with $n \geq n_0$.
- **Many sequences can be a solution for the same recurrence relation.**

$$a_n = 2a_{n-1} - a_{n-2}, \text{ for } n \geq 2$$

The following sequences are solutions of this recurrence relation:

# Recurrence Relations

- 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 $a_0, a_1, \ldots, a_{n-1}$, for all integers $n$ with $n \geq n_0$.
- **Many sequences can be a solution for the same recurrence relation.**

$$a_n = 2a_{n-1} - a_{n-2}, \text{ for } n \geq 2$$

The following sequences are solutions of this recurrence relation:

  - $a_n = 3n$, for all $n \geq 0$,

# Recurrence Relations

- 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 $a_0, a_1, \ldots, a_{n-1}$, for all integers $n$ with $n \geq n_0$.
- **Many sequences can be a solution for the same recurrence relation.**

$$a_n = 2a_{n-1} - a_{n-2}, \text{ for } n \geq 2$$

The following sequences are solutions of this recurrence relation:

  - $a_n = 3n$, for all $n \geq 0$,
  - $a_n = 5$, for all $n \geq 0$.

# Recurrence Relations

- 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 $a_0, a_1, \ldots, a_{n-1}$, for all integers $n$ with $n \geq n_0$.
- **Many sequences can be a solution for the same recurrence relation.**

  $$a_n = 2a_{n-1} - a_{n-2}, \text{ for } n \geq 2$$

  The following sequences are solutions of this recurrence relation:
  - $a_n = 3n$, for all $n \geq 0$,
  - $a_n = 5$, for all $n \geq 0$.
- The **initial conditions** for a sequence specify the terms before $n_0$ (before the recurrence relation takes effect).
  The recurrence relations together with the initial conditions uniquely determines the sequence. For the example above, the initial conditions are: $a_0 = 0$, $a_1 = 3$; and $a_0 = 5$, $a_1 = 5$; respectively.

# Modeling with Recurrence Relations (used for advanced counting)

- Compound interest: A person deposits \$10,000 into savings that yields 11% per year with interest compound annually. How much is in the account in 30 years?
- Growth of rabbit population on an island:
  A young pair of rabbits of opposite sex are placed on an island. A pair of rabbits do not breed until they are 2 months old, but then they produce another pair each month. Find a recurrence relation for the number of pairs of rabbits on the island after $n$ months.
- The Hanoi Tower:
  Setup a recurrence relation for the sequence representing the number of moves needed to solve the Hanoi tower puzzle.
- Find a recurrence relation for the number of bit strings of length $n$ that do not have two consecutive 0s, and also give initial conditions.

# Linear Homogeneous Recurrence Relations

We will study more closely **linear homogeneous recurrence relations of degree $k$ with constant coefficients**:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

where $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

linear = previous terms appear with exponent $1$ (not squares, cubes, etc),
homogeneous = no term other than the multiples of $a_i$'s,
degree $k$= expressed in terms of previous $k$ terms
constant coefficients = coefficients in front of the terms are constants, instead of general functions.

This recurrence relation plus $k$ initial conditions uniquely determines the sequence.

Which of the following are linear homogeneous recurrence relations of degree $k$ with constant coefficients? If yes, determine $k$; if no, explain why not.

- $P_n = (1.11)P_{n-1}$
- $f_n = f_{n-1} + f_{n-2}$
- $H_n = 2H_{n-1} + 1$
- $a_n = a_{n-5}$
- $a_n = a_{n-1} + a_{n-2}^2$
- $B_n = nB_{n-1}$

# Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

## Theorem (1)

*Let $c_1$ and $c_2$ be real numbers. Suppose that $r^2 - c_1 r - c_2 = 0$ has two distinct roots $r_1$ and $r_2$. Then, the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \ldots$, where $\alpha_1$ and $\alpha_2$ are constants.*

Proof: ($\Leftarrow$) If $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, then $\{a_n\}$ is a solution for the recurrence relation.

($\Rightarrow$) If $\{a_n\}$ is a solution for the recurrence relation, then $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, for some constants $\alpha_1$ and $\alpha_2$.

**Exercises:**

1. Solve: $a_n = a_{n-1} + 2a_{n-2}$ with $a_o = 2$ and $a_1 = 7$
2. Find explicit formula for the Fibonacci Numbers.

Root with multiplicity 2...

## Theorem (2)

*Let $c_1$ and $c_2$ be real numbers with $c_2 \neq 0$. Suppose that $r^2 - c_1 r - c_2 = 0$ has only one root $r_0$. A sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$, for $n = 0, 1, 2 \ldots$, where $\alpha_1$ and $\alpha_2$ are constants.*

**Exercise:**

Solve the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$, with initial conditions $a_0 = 1$, $a_1 = 6$.

**Exercise:**
Solve the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$, with initial conditions $a_0 = 1$, $a_1 = 6$.

**Solution:**
$r^2 - 6r + 9 = 0$ has only $3$ as a root.
So the format of the solution is $a_n = \alpha_1 3^n + \alpha_2 n 3^n$. Need to determine $\alpha_1$ and $\alpha_2$ from initial conditions:

$$\begin{aligned} a_0 &= 1 = \alpha_1 \\ a_1 &= 6 = \alpha_1 \cdot 3 + \alpha_2 3 \end{aligned}$$

Solving these equations we get $\alpha_1 = 1$ and $\alpha_2 = 1$.
Therefore, $a_n = 3^n + n3^n$.

**Question: how can you double check this answer is right?**

## Theorem (3)

*Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose that the characteristic equation $r^k - c_1 r^{k-1} - \cdots - c_k = 0$ has $k$ distinct roots $r_1, r_2, \ldots, r_k$. Then, a sequence $\{a_n\}$ is a solution of the recurrence relation*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

*if and only if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$ for $n = 0, 1, 2, \ldots,$ where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are constants.*

**Exercise:**

Find the solution to the recurrence relation

$$a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3},$$

with the initial conditions $a_0 = 2$, $a_1 = 5$ ,and $a_2 = 15$.

## Theorem (4)

*Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose that the characteristic equation $r^k - c_1 r^{k-1} - \cdots - c_k = 0$ has $t$ distinct roots $r_1, r_2, \ldots, r_t$ with multiplicities $m_1, m_2, \ldots, m_t$, respectively, so that $m_i \geq 1$ and $m_1 + m_2 + \cdots + m_t = k$. Then, a sequence $\{a_n\}$ is a solution of the recurrence relation*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

*if and only if*

$$\begin{aligned} a_n &= (\alpha_{1,0} + \alpha_{1,1} n + \cdots \alpha_{1,m_1-1} n^{m_1-1}) r_1^n \\ &\quad + (\alpha_{2,0} + \alpha_{2,1} n + \cdots \alpha_{2,m_2-1} n^{m_2-1}) r_2^n \\ &\quad + \cdots + (\alpha_{t,0} + \alpha_{t,1} n + \cdots \alpha_{t,m_t-1} n^{m_t-1}) r_t^n \end{aligned}$$

*for $n = 0, 1, 2, \ldots,$ where $\alpha_{i,j}$ are constants for $1 \leq i \leq t$, $0 \leq j \leq m_i - 1$ .*

**Exercise:**

Find the solution to the recurrence relation

$$a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3},$$

with initial conditions $a_0 = 1$, $a_1 = -2$ and $a_2 = -1$.

# Non-homogeneous Recurrence Relations

We look not at **linear non-homogeneous recurrence relation with constant coefficients**, that is, one of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

where $c_1, c_2, \ldots, c_k$ are real numbers and $F(n)$ is a function not identically zero depending only on $n$.
The recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

is the **associated homogeneous recurrence relation**.

# Solving Non-homogeneous Linear Recurrence Relations

## Theorem (5)

*If $\{a_n^{(p)}\}$ is a particular solution for the non-homogeneous linear recurrence relation with constant coefficients*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

*then every solution is of the form $\{a_n^{(p)} + a_n^{(h)}\}$, where $\{a_n^{(h)}\}$ is a solution of the associated homogeneous recurrence relation*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

Key: find a particular solution to the non-homogeneous case and we are done, since we know how to solve the homogeneous one.

# Finding a particular solution

## Theorem (6)

*Suppose that* $\{a_n\}$ *satisfies the linear non-homogeneous recurrence relation* $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n)$, *where* $c_1, c_2, \ldots, c_k$ *are real numbers and* $F(n) = (b_t n^t + b_{t-1} n^{t-1} + \cdots + b_1 n + b_0) s^n$, *where* $b_0, b_1, \ldots, b_t$ *and* $s$ *are real numbers.*

*When* $s$ *is NOT a root of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form*

$$(p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n.$$

*When* $s$ *is a root of the characteristic equation and its multiplicity is* $m$, *there is a particular solution of the form*

$$n^m (p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n.$$

**Exercises:** (roots of characteristic polynomial are given to simplify your work)
Find all solutions of

- $a_n = 3a_{n-1} + 2n$. What is the solution with $a_1 = 3$? (root: $r_1 = 3$)
- $a_n = 5a_{n-1} - 6a_{n-2} + 7^n$ (root: $r_1 = 3, r_2 = 2$)

What is the form of a particular solution to

$$a_n = 6a_{n-1} - 9a_{n-2} + F(n),$$

when:

- $F(n) = 3^n$,
- $F(n) = n3^n$,
- $F(n) = n^2 2^n$,
- $F(n) = (n^2 + 1)3^n$.

(root: $r_1 = 3$, multiplicity 2)

# Divide-and-Conquer Recurrence Relations

- Divide-and-conquer algorithms:

# Divide-and-Conquer Recurrence Relations

- Divide-and-conquer algorithms:
  - divide a problem of size $n$ into $a$ subproblems of size $b$,

# Divide-and-Conquer Recurrence Relations

- Divide-and-conquer algorithms:
  - divide a problem of size $n$ into $a$ subproblems of size $b$,
  - use some extra operations to combine the individual solutions into the final solution for the problem of size $n$, say $g(n)$ steps.

# Divide-and-Conquer Recurrence Relations

- Divide-and-conquer algorithms:
  - divide a problem of size $n$ into $a$ subproblems of size $b$,
  - use some extra operations to combine the individual solutions into the final solution for the problem of size $n$, say $g(n)$ steps.
- Examples: binary search, merge sort, fast multiplication of integers, fast matrix multiplication.

# Divide-and-Conquer Recurrence Relations

- Divide-and-conquer algorithms:
  - divide a problem of size $n$ into $a$ subproblems of size $b$,
  - use some extra operations to combine the individual solutions into the final solution for the problem of size $n$, say $g(n)$ steps.
- Examples: binary search, merge sort, fast multiplication of integers, fast matrix multiplication.
- A divide-and-conquer recurrence relation, expresses the number of steps $f(n)$ needed to solve the problem:

$$f(n) = af(n/b) + cn^d.$$

(for simplicity assume this is defined for $n$ that are multiples of $b$; otherwise there are roundings up or down to closest integers)

# Examples

Give the recurrence relations for:

- Mergesort
- Binary search
- Finding both maximum and minimum over a array of length $n$ by dividing it into 2 pieces and the comparing their individual maxima and minima.

# Master Theorem for Divide-and-Conquer Recurrence Relations

## Theorem (Master Theorem)

*Let $f$ be an increasing function that satisfies the recurrence relation:*

$$f(n) = af(n/b) + cn^d,$$

*whenever $n = b^k$, where $k$ is a positive integer, $a \geq 1$, $b$ is an integer greater than $1$, and $c$ and $d$ are real numbers with $c$ positive and $d$ non-negative. Then,*

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

# Proof of the master theorem

We can prove the theorem by showing the following steps:

1. Show that if $a = b^d$ and $n$ is a power of $b$, then $f(n) = f(1)n^d + cn^d \log_b n$.
   Once this is shown, it is clear that if $a = b^d$ then $f(n) \in O(n^d \log n)$.
2. Show that if $a \neq b^d$ and $n$ is a power of $b$, then $f(n) = c_1 n^d + c_2 n^{log_b a}$, where $c_1 = b^d c/(b^d - a)$ and $c_2 = f(1) + b^d c/(a - b^d)$.
3. Once the previous is shown, we get:
   if $a < b^d$, then $\log_b a < d$, so $f(n) = c_1 n^d + c_2 n^{\log_b a} \leq (c_1 + c_2)n^d \in O(n^d)$.
   if $a > b^d$, then $\log_b a > d$, so $f(n) = c_1 n^d + c_2 n^{\log_b a} \leq (c_1 + c_2)n^{\log_b a} \in O(n^{\log_b a})$.

# Proving item 1:

## Lemma

*If $a = b^d$ and $n$ is a power of $b$, then $f(n) = f(1)n^d + cn^d \log_b n$.*

**Proof:**

Let $k = \log_b n$, that is $n^k = b$. Iterating $f(n) = af(n/b) + cn^d$, we get:

$$\begin{aligned}
f(n) &= a(af(n/b^2) + c(n/b)^d) + cn^d = a^2 f(n/b^2) + ac(n/b)^d + cn^d \\
&= a^2(af(n/b^3) + c(n/b^2)) + ac(n/b)^d + cn^d \\
&= a^3 f(n/b^3) + a^2 c(n/b^2)^d + ac(n/b)^d + cn^d \\
&= \ldots = a^k f(1) + \sum_{j=0}^{k-1} a^j c(n/b^j)^d = a^k f(1) + \sum_{j=0}^{k-1} cn^d \\
&= a^k f(1) + kcn^d = a^{\log_b n} f(1) + (\log_b n)cn^d \\
&= n^{\log_b a} f(1) + cn^d \log_b n = n^d f(1) + cn^d \log_b n.
\end{aligned}$$

# Proving item 2:

## Lemma

*If $a \neq b^d$ and $n$ is a power of $b$, then $f(n) = c_1 n^d + c_2 n^{log_b a}$, where $c_1 = b^d c/(b^d - a)$ and $c_2 = f(1) + b^d c/(a - b^d)$.*

**Proof:**

Let $k = \log_b n$; i. e. $n = b^k$. We will prove the lemma by induction on $k$.

Basis: If $n = 1$ and $k = 0$, then

$c_1 n^d + c_2 n^{\log_b a} = c_1 + c_2 = b^d c/(b^d - a) + f(1) + b^d c/(a - b^d) = f(1)$.

Inductive step: Assume lemma is true for $k$, where $n = b^k$. Then, for $n = b^{k+1}$, $f(n) = af(n/b) + cn^d =$

$a((b^d c/(b^d - a))(n/b)^d + (f(1) + b^d c/(a - b^d))(n/b)^{\log_b a})) + cn^d =$

$(b^d c/(b^d - a))n^d a/b^d + (f(1) + b^d c/(a - b^d))n^{\log_b a} + cn^d =$

$n^d[ac/(b^d - a) + c(b^d - a)/(b^d - a)] + [f(1) + b^d c/(a - b^d c)]n^{\log_b a} =$

$(b^d c/(b^d - a))n^d + (f(1) + b^d c/(a - b^d))n^{\log_b a}$. □

Use the master theorem to determine the asymptotic growth of the following recurrence relations:

- binary search: $b(n) = b(n/2) + 2$;
- mergesort: $M(n) = 2M(n/2) + n$;
- maximum/minima: $m(n) = 2m(n/2) + 2$.

You have divided and conquered; have you saved in all cases?

# Elements of Graph Theory

**Quick review of Chapters 9.1… 9.5, 9.7 (studied in Mt1348/2008) = all basic concepts must be known**

**New topics**

- we will mostly skip shortest paths (Chapter 9.6), as that was covered in Data Structures
- Graph colouring (Chapter 9.8)
- Trees (Chapter 3.1, 3.2)

# Applications of Graphs

Applications of Graphs: Potentially anything (graphs can represent relations, relations can describe the extension of any predicate).

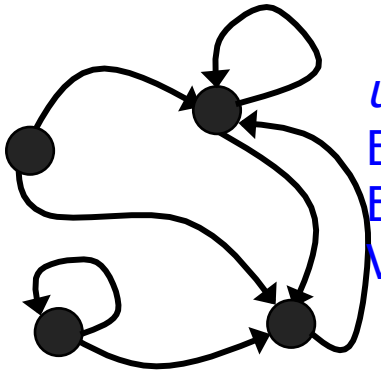
Applications in networking, scheduling, flow optimization, circuit design, path planning.

More applications: Geneology analysis, computer game-playing, program compilation, object-oriented design, ...

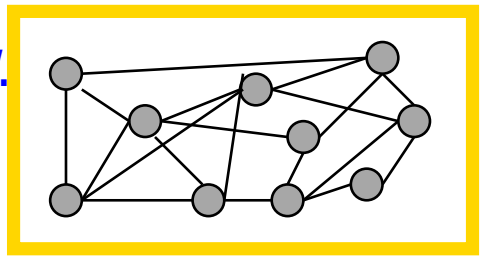
# Simple Graphs

Simple Graphs: Correspond to symmetric, irreflexive binary relations *R*.

- A *simple graph* $G=(V,E)$ consists of:
  - a set *V* of *vertices* or *nodes* (*V* corresponds to the universe of the relation *R*),
  - a set *E* of *edges* / *arcs* / *links*: unordered pairs of [distinct] elements $u,v \in V$, such that $uRv$.
- A *directed graph* (*V*,*E*) consists of a set of vertices *V* and a binary relation (need not be symmetric) *E* on *V*.



*u*, *v* are *adjacent* / *neighbors* / *connected*.
Edge *e* is *incident with* vertices *u* and *v*.
Edge *e connects u* and *v*.
Vertices *u* and *v* are *endpoints* of edge *e*.



*Visual Representation of a Simple Graph*

# Degree of a Vertex

- Let $G$ be an undirected graph, $v \in V$ a vertex.
- The *degree* of $v$, $\deg(v)$, is its number of incident edges. (Except that any self-loops are counted twice.)
- A vertex with degree 0 is called *isolated*.
- A vertex of degree 1 is called *pendant*.

Handshaking Theorem: Let $G$ be an undirected (simple, multi-, or pseudo-) graph with vertex set $V$ and edge set $E$. Then

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

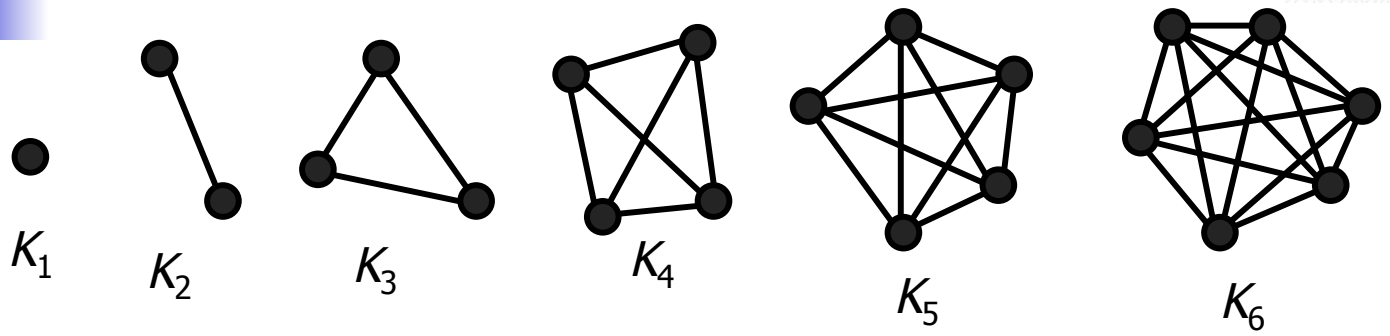
**Corollary:** Any undirected graph has an even number of vertices of odd degree.

# Directed Degree

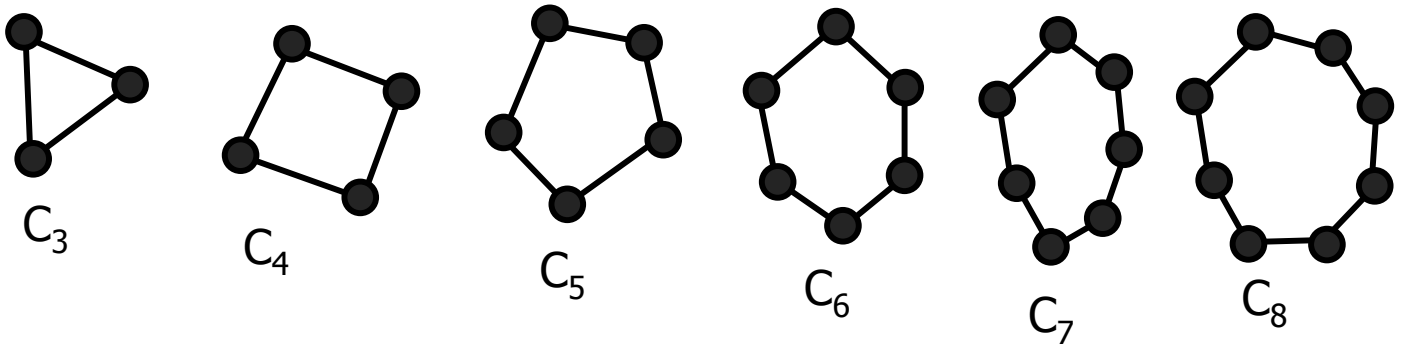
- Let *G* be a directed graph, *v* a vertex of *G*.
  - The *in-degree* of *v*, $\deg^-(v)$, is the number of edges going to *v*.
  - The *out-degree* of *v*, $\deg^+(v)$, is the number of edges coming from *v*.
  - The *degree* of *v*, $\deg(v) := \deg^-(v) + \deg^+(v)$, is the sum of *v*'s in -degree and out-degree.
- Directed Handshaking Theorem:

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = \frac{1}{2} \sum_{v \in V} \deg(v) = |E|$$

# Special Graph Structures

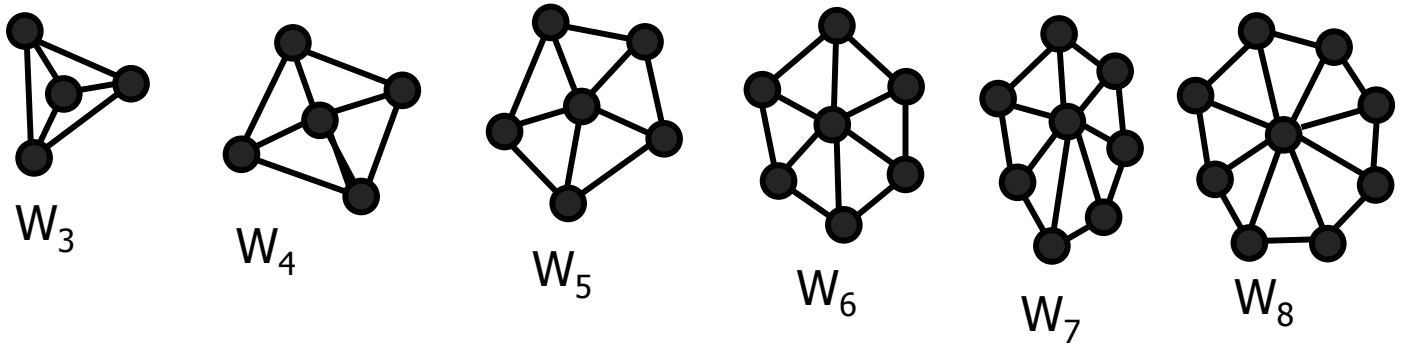



Complete graphs $K_n$

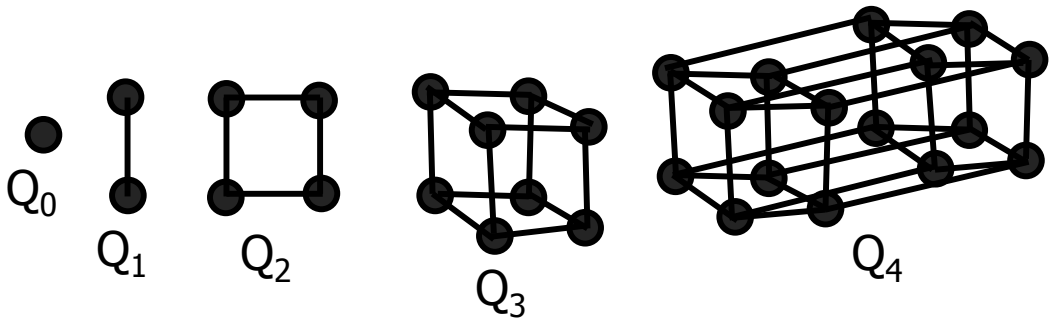



Cycles $C_n$

# Special Graph Structures




Wheels $W_n$

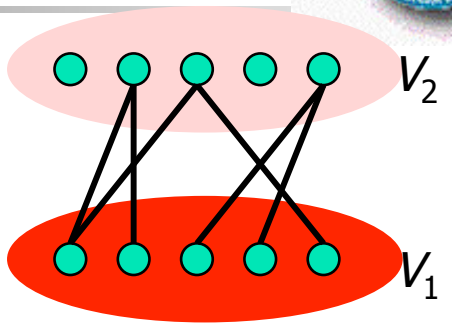



*n*-Cubes $Q_n$

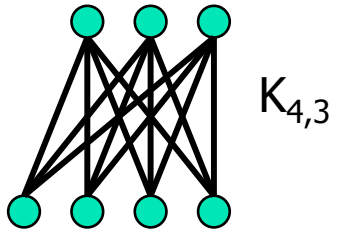
*Number of vertices:* $2^n$. *Number of edges?*

# Bipartite Graphs

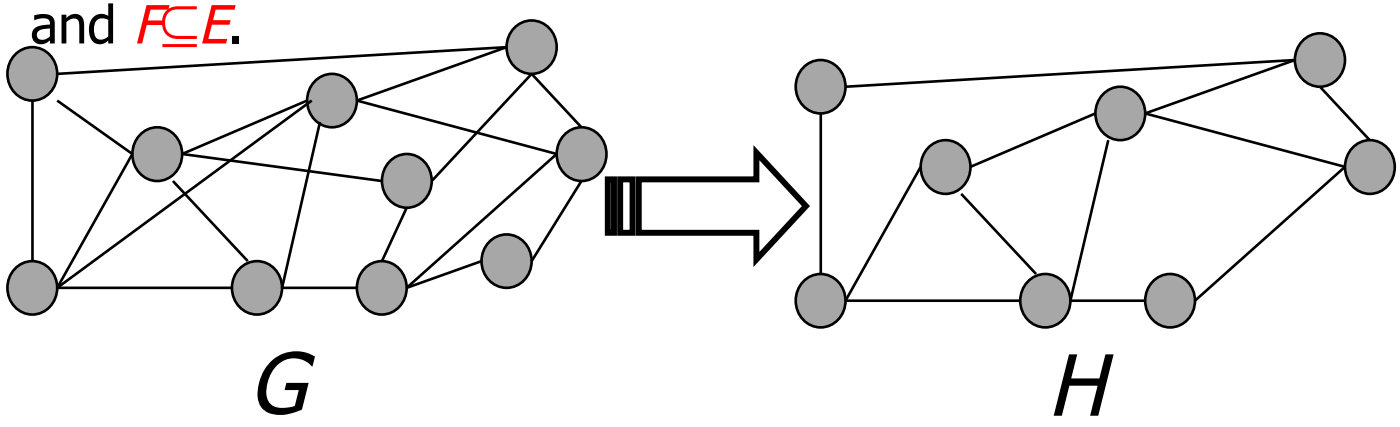
A graph $G=(V,E)$ is *bipartite* (two-part) iff $V = V_1 \cap V_2$ where $V_1 \cup V_2 = \varnothing$ and $\forall e \in E$: $\exists v_1 \in V_1, v_2 \in V_2$: $e=\{v_1,v_2\}$.




For $m,n \in \mathbf{N}$, the *complete bipartite graph* $K_{m,n}$ is a bipartite graph where $|V_1| = m$, $|V_2| = n$, and $E = \{\{v_1,v_2\} | v_1 \in V_1 \wedge v_2 \in V_2\}$.




A subgraph of a graph $G=(V,E)$ is a graph $H=(W,F)$ where $W \subseteq V$ and $F \subseteq E$.

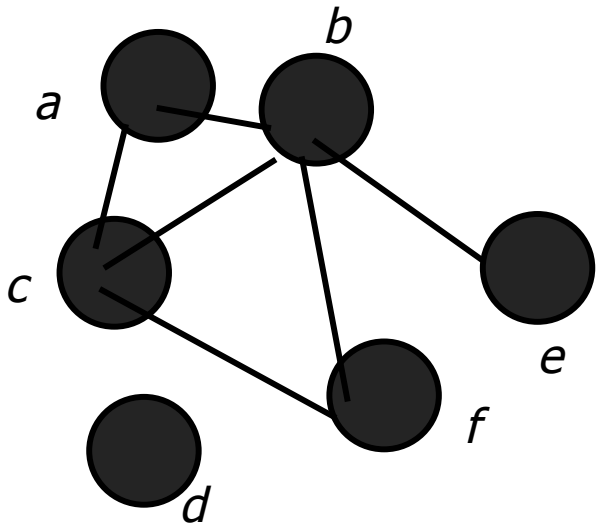


# §9.3: Graph Representations & Isomorphism

- Graph representations:
  - Adjacency lists.
  - Adjacency matrices.
  - Incidence matrices.
- Graph isomorphism:
  - Two graphs are isomorphic iff they are identical except for their node names.

# Adjacency Lists

Adjacency Lists: A table with 1 row per vertex, listing its adjacent vertices.




| Vertex | Adjacent Vertices |
|---|---|
| $a$ | $b, c$ |
| $b$ | $a, c, e, f$ |
| $c$ | $a, b, f$ |
| $d$ | |
| $e$ | $b$ |
| $f$ | $c, b$ |

Directed Adjacency Lists: 1 row per node, listing the terminal nodes of each edge incident from that node.

# Adjacency Matrices

- A way to represent simple graphs
  - possibly with self-loops.
- Matrix $\mathbf{A}=[a_{ij}]$, where $a_{ij}$ is 1 if $\{v_i, v_j\}$ is an edge of $G$, and is 0 otherwise.
- Can extend to pseudographs by letting each matrix elements be the number of links (possibly >1) between the nodes.

# Graph Isomorphism

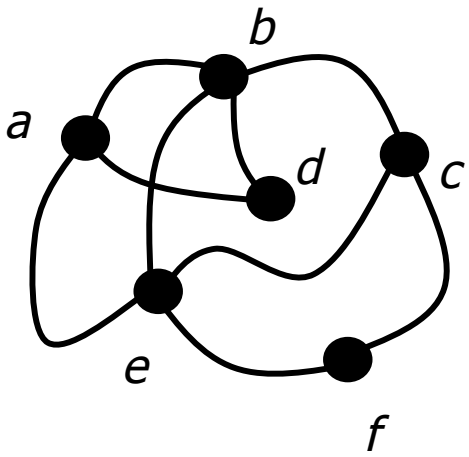
- Formal definition:
  - Simple graphs $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ are *isomorphic* iff ∃ a bijection $f: V_1 \rightarrow V_2$ such that ∀ $a,b \in V_1$, $a$ and $b$ are adjacent in $G_1$ iff $f(a)$ and $f(b)$ are adjacent in $G_2$.
  - $f$ is the "renaming" function between the two node sets that makes the two graphs identical.
  - This definition can easily be extended to other types of graphs.

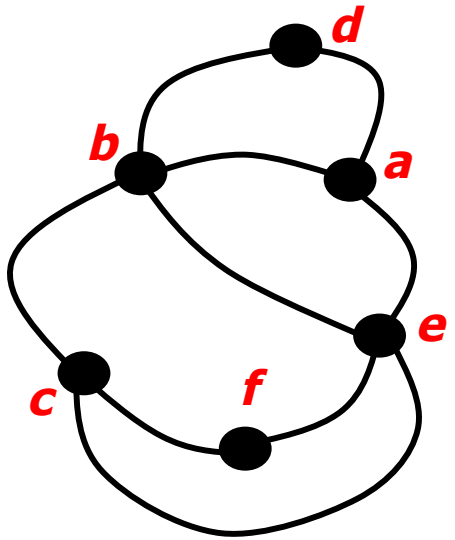
*Necessary* but not *sufficient* conditions for $G1=(V1, E1)$ to be isomorphic to $G2=(V2, E2)$:

- We must have that $|V1|=|V2|$, and $|E1|=|E2|$.
- The number of vertices with degree $n$ is the same in both graphs.
- For every proper subgraph $g$ of one graph, there is a proper subgraph of the other graph that is isomorphic to $g$.

# Isomorphism Example

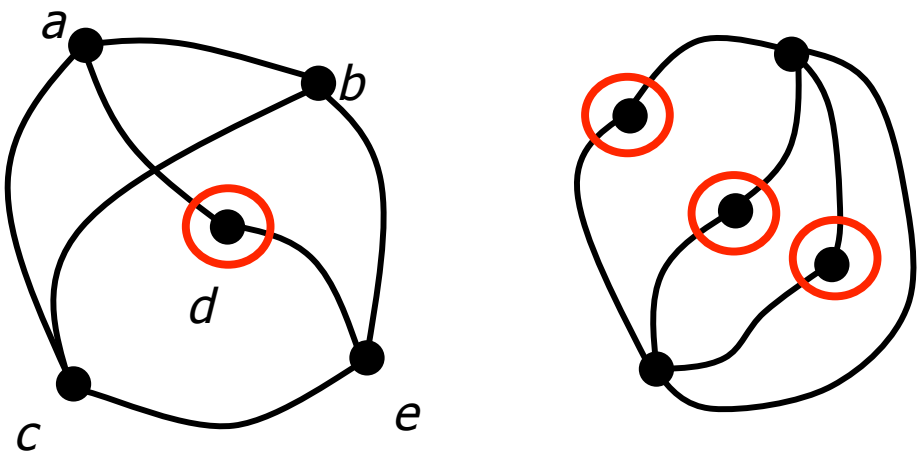
If isomorphic, label the 2nd graph to show the isomorphism, else identify difference.



# Are These Isomorphic?

- If isomorphic, label the 2nd graph to show the isomorphism, else identify difference.




- *Same # of vertices*
- *Same # of edges*
- *Different # of vertices of degree 2! (1 vs 3)*

# §9.4: Connectivity

- In an undirected graph, a *path of length n from u to v* is a sequence of adjacent edges going from vertex *u* to vertex *v*.
- A path is a *circuit* if *u=v*.
- A path *traverses* the vertices along it.
- A path is *simple* if it contains no edge more than once.

- Paths in Directed Graphs: Same as in undirected graphs, but the path must go in the direction of the arrows.

An undirected graph is *connected* iff there is a path between every pair of distinct vertices in the graph.

There is a *simple* path between any pair of vertices in a connected undirected graph.

*Connected component*: connected subgraph

A *cut vertex* or *cut edge* separates 1 connected component into 2 if removed

# Directed Connectedness

- A directed graph is *strongly connected* iff there is a directed path from *a* to *b* for any two vertices *a* and *b*.
- It is *weakly connected* iff the underlying *undirected* graph (*i.e.*, with edge directions removed) is connected.
- Note *strongly* implies *weakly* but not vice-versa.

Note that connectedness, and the existence of a circuit or simple circuit of length *k* are graph invariants with respect to isomorphism.

Counting different paths: the number of different paths from a vertex *i* to a vertex *j* is the (i, j) entry in $\mathbf{A}^r$, where A is the adjacency matrix of the graph

proof by induction on *r*

# §9.5: Euler & Hamilton Paths

- An ***Euler circuit*** in a graph *G* is a simple circuit containing every edge of *G*.
- An ***Euler path*** in *G* is a simple path containing every edge of *G*.
- A ***Hamilton circuit*** is a circuit that traverses each vertex in *G* exactly once.
- A ***Hamilton path*** is a path that traverses each vertex in *G* exactly once.

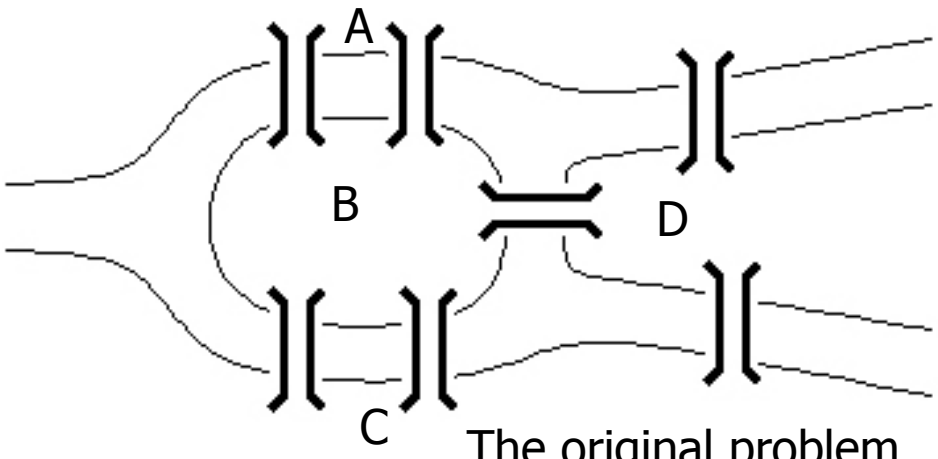
# Euler and Hamiltonian Tours

**Chinese postman problem**

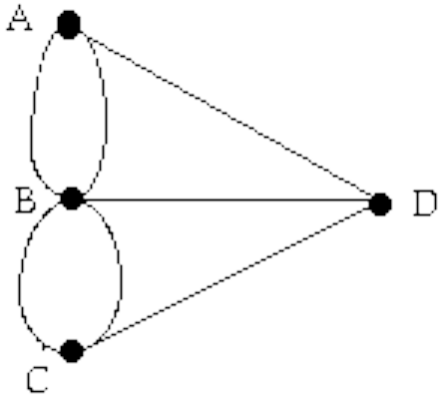
- find a shortest tour in a non-Euler graph
- some edges will be traversed twice
- corresponds to finding the cheapest set of paths connecting matching vertices of odd degree
    - the number of odd-degree vertices is even
    - the paths are edge-disjoint

# Bridges of Königsberg Problem

Can we walk through town, crossing each bridge exactly once, and return to start?




The original problem




Equivalent multigraph

**Theorem:** A connected multigraph has an Euler circuit iff each vertex has even degree.

**Proof:**

(→) The circuit contributes 2 to degree of each node.

(←) By construction using algorithm on p. 580-581

# Euler Path Problem

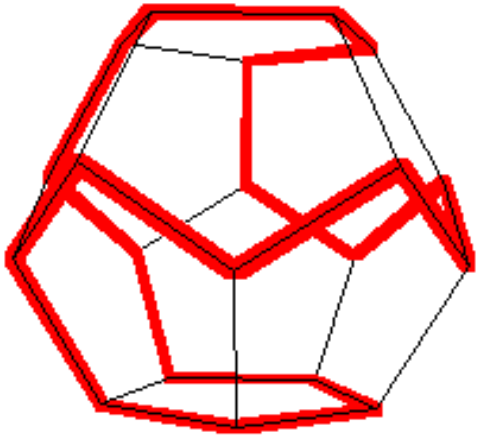
- **Theorem:** A connected multigraph has an Euler path (but not an Euler circuit) iff it has exactly 2 vertices of odd degree.
  - One is the start, the other is the end.

- **Euler tour in a directed graph**
  - in-degrees must match out-degrees in all nodes

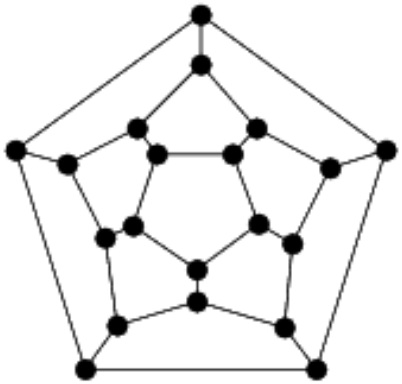
- Euler Circuit Algorithm
  - Begin with any arbitrary node.
  - Construct a simple path from it till you get back to start.
  - Repeat for each remaining subgraph, splicing results back into original cycle.

# Round-the-World Puzzle

- Can we traverse all the vertices of a dodecahedron, visiting each once?`



Dodecahedron puzzle



Equivalent graph

# Hamiltonian Path Theorems

- **Dirac's theorem**: If (but not only if) $G$ is connected, simple, has $n \geq 3$ vertices, and $\forall v \deg(v) \geq n/2$, then $G$ has a Hamilton circuit.
  - **Ore's corollary:** If $G$ is connected, simple, has $n \geq 3$ nodes, and $\deg(u)+\deg(v) \geq n$ for every pair $u,v$ of non-adjacent nodes, then $G$ has a Hamilton circuit.

# Hamiltonian Tours - Applications

**Traveling salesmen problem**

- in a weighted graph, find the shortest tour visiting every vertex
- we can solve it if we can solve the problem of finding the shortest Hamiltonian path in complete graphs

**Gray codes**

- find a sequence of codewords such that each binary string is used, but adjacent codewords are close to each other (differ by 1 bit only)
- all binary strings of length **n** = vertices of **n-**dimensional hypercube
- edges of the hypercube = vertices that differ by 1 bit
- our problem = find a Hamiltonian circuit in hypercubes
- Gray codes – one particular solution
    - can be defined recursively (as hypercubes are)

# Planar Graphs

***Planar graphs*** are graphs that can be drawn in the plane without edges having to cross.

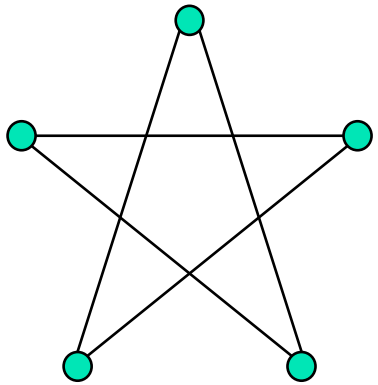
Understanding planar graph is important:

- Any graph representation of maps/ topographical information is planar.
  - graph algorithms often specialized to planar graphs (e.g. traveling salesperson)
- Circuits usually represented by planar graphs

# Planar Graphs
# -Common Misunderstanding

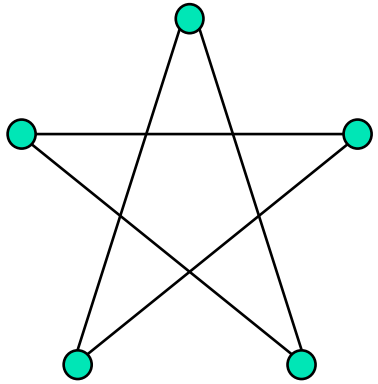
Just because a graph is drawn with edges crossing doesn't mean its not planar.

Q: Why can't we conclude that the following is non-planar?



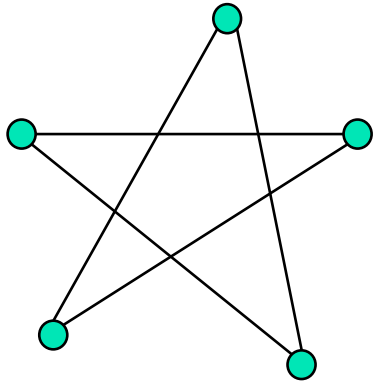
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



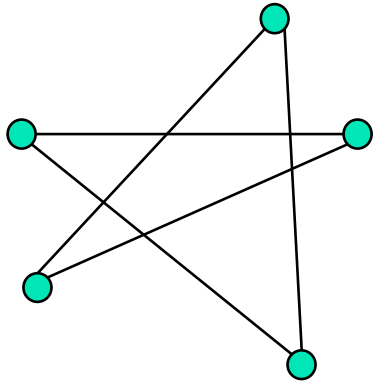
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



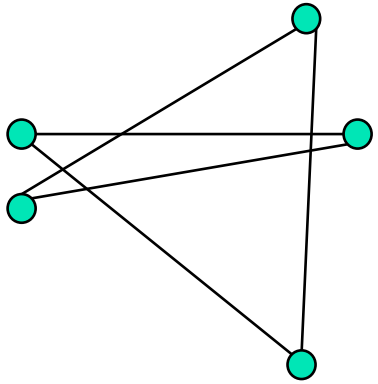
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



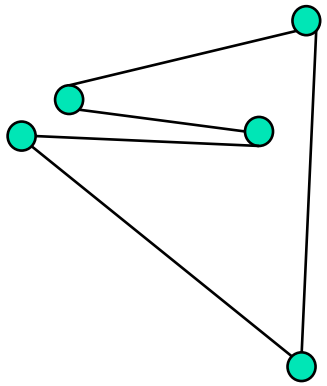
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



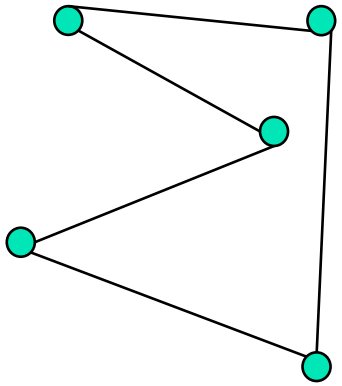
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



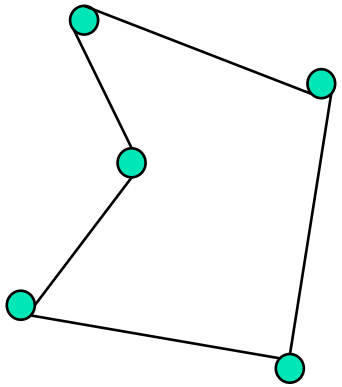
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



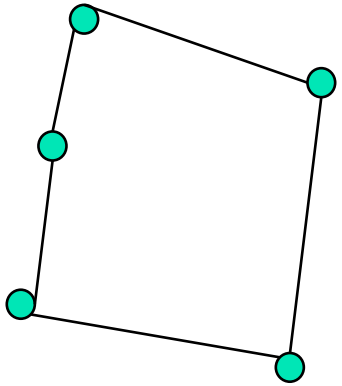
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



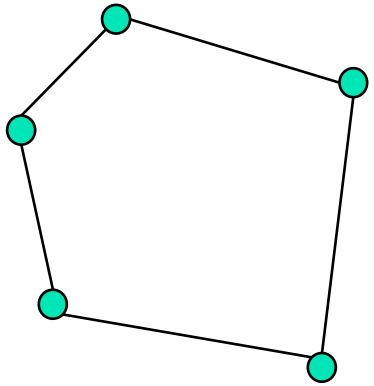
# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:



# Planar Graphs
# -Common Misunderstanding

A: Because it is isomorphic to a graph which *is* planar:

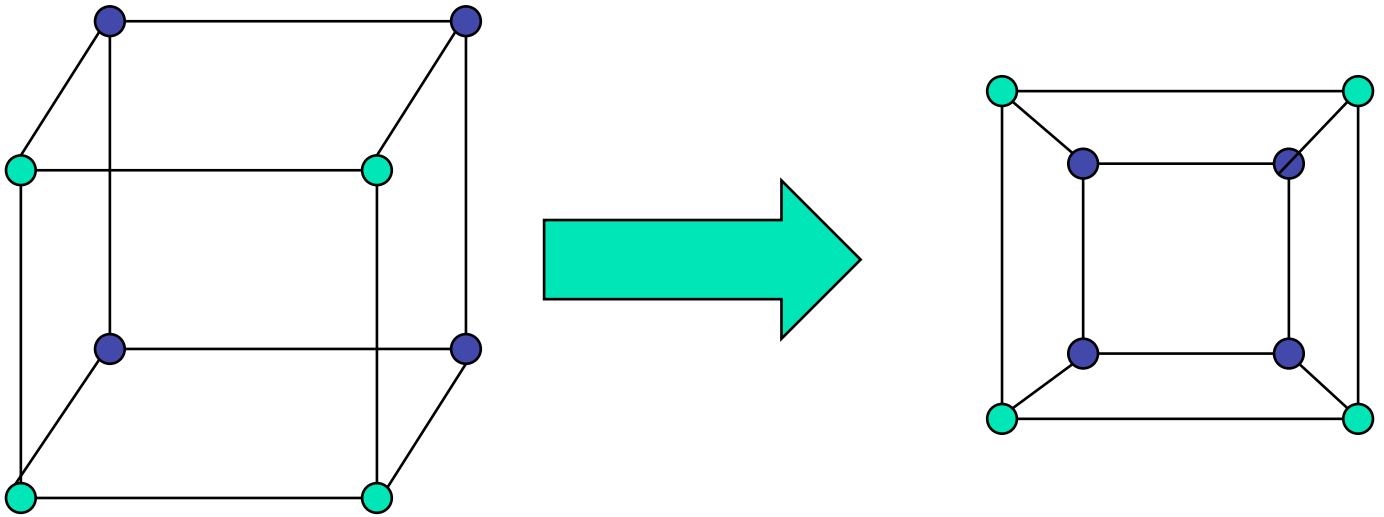


# Proving Planarity

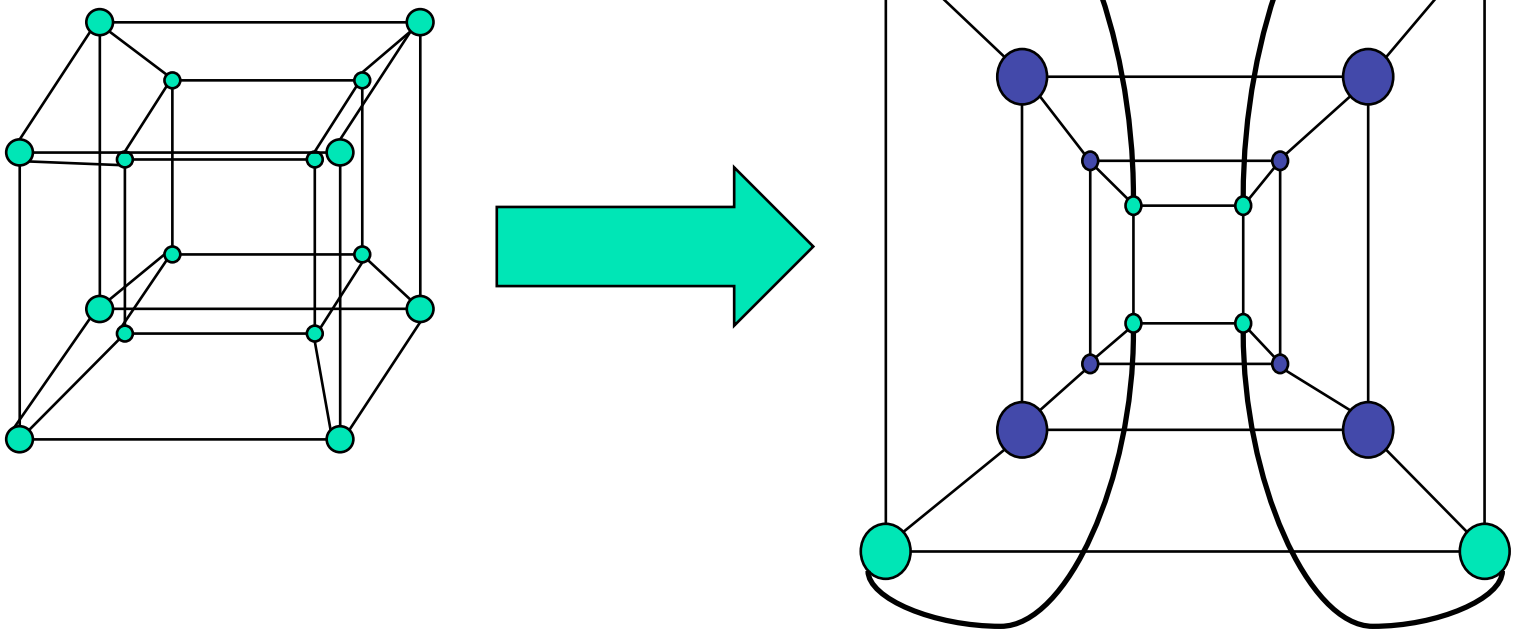
To prove that a graph is planar amounts to redrawing the edges in a way that no edges will cross. May need to move vertices around and the edges may have to be drawn in a very indirect fashion.

E.G. show that the 3-cube is planar:

# Proving Planarity
# 3-Cube



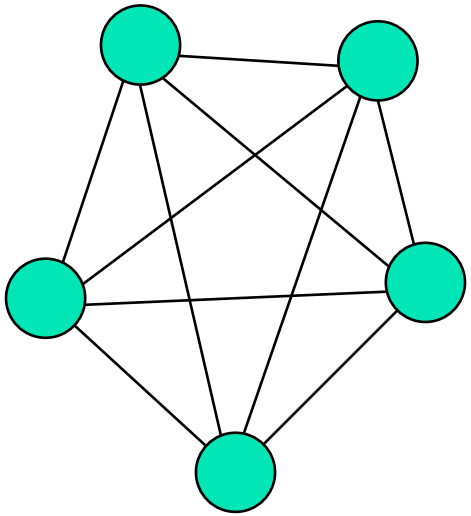
# Proving Planarity?
# 4-Cube

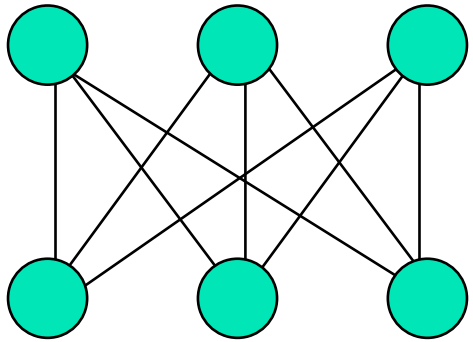


Seemingly not planar, but how would one prove this!

# The smallest graphs that are not planar

- $K_5$, $K_{3,3}$





## Disproving Planarity: Kuratowski / Wagner

- A graph is planar if and only if it does not contain the $K_5$ and the $K_{3,3}$ as a *homeomorphic subgraph* / as a *minor.*
- Minor: H is a minor of G, if H can be obtained from G by a series of 0 or more deletions of vertices, deletions of edges, and contraction of edges.
- Does not yield fast recognition algorithm!

# Euler's theorem

- Let G be a connected plane graph with $v$ vertices, $e$ edges, and $f$ faces.

  Then $v + f - e = 2$.

- Proof. By induction.
  - True if $e$=0.
  - If G has a circuit, then delete an edge and …
  - If G has a vertex $w$ of degree 1, then delete $w$ and …
- …

# Euler's theorem Corollaries

- If G is a connected plane graph with no parallel edges and no self-loops, with **$v \geq 3$** , then **$e \leq 3v-6$**.
  - Every face `has' at least three edges; each edge `is on' two faces, or twice on the same face.
- Every plane graph with no parallel edges and no self -loops has a vertex of degree at most 5.
  - **$K_5$** is not a planar graph
- If **G** is a planar simple graph with **$v \geq 3$** and no cycles of length **3**, then **$e \leq 2v-4$**
  - **$K_{3,3}$** is not planar

# Proof of Euler's theorem Corollaries

Suppose that G has at most m edges, consider some plane drawing of G, with f faces. Consider the number of pairs (e, F) where e is one of the edges bounding the face F.

For each edge e, there are at most 2 faces that it bounds. So the total number of these edge face pairs has to be less than 2e. On the other hand, because G is a simple graph, each face is bounded by at least 3 edges. Therefore, the total number of edge-face pairs is greater than or equal to 3f. So $3f \leq 2e$

By the Euler Polyhedron Formula, v-e+f=2, so, 3v-3e+3f=6. Since $3f \leq 2e$, $3f = 6 - 3v + 3e \leq 2e$. Therefore $e \leq 3v - 6$.

If G has no triangles, each face must be bounded by 4 or more edges. Thus $2f \leq e$ and $4 - 2v + 2e \leq e$ , therefore $e \leq 2v - 4$.

# Euler's theorem Corollaries

Every simple, planar graph has a vertex of degree less than 6.

Proof: $\sum_{w \in V} \deg(w) = 2e \leq 2(3v - 6) = 6v - 12$

Thus the average degree <=(6v-12)/v = 6 – 12/v < 6.

So at least one of the vertices has degree less than 6.

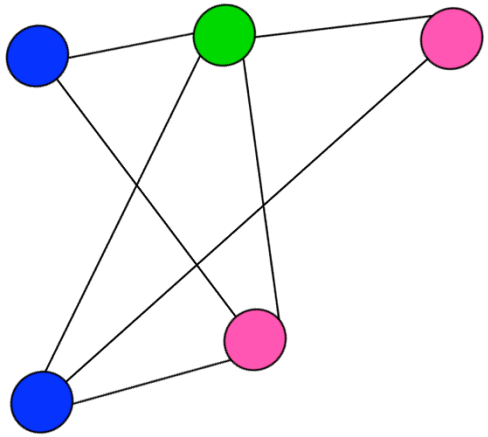
# Graph Colorings

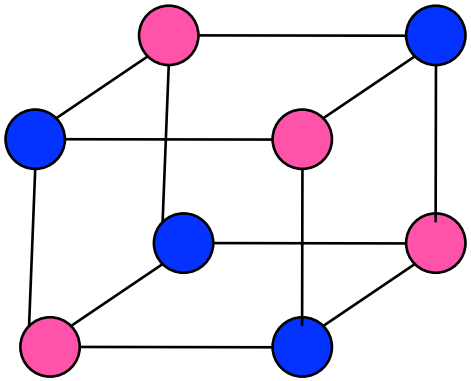
## Vertex coloring of a graph

- assign a color to each vertex so that adjacent vertices are of different colors
- i.e. find **c: V → N** such that **(u,v) ∈E→ c(u) ≠ c(v)**

## Chromatic number $\chi$ of a graph G

- the least amount of colors needed to color the graph





# Graph Colorings

So, what is a chromatic number for

- $K_n$?
- $C_n$?
- $K_{m,n}$?

Bipartite Graphs

The chromatic number of a graph G is 2 if and only if G is a bipartite graph

Planar Graphs?

# The four Color Theorem

**The four color theorem: The chromatic number of every simple planar graph is at most four**

We can prove that six  colors are enough

For general graphs?

- only exponential algorithms known
- even finding approximation is difficult

# Six color Theorem

Proof of the six color theorem: by induction on n, the number of vertices of the graph.

Basis Step: If G has fewer than seven vertices then the result is obvious.

Inductive step: Let n>=7. We assume that all simple graphs with n-1 vertices are 6 colorable. Because of planarity and Euler's theorem we know that G has a vertex v with degree less than 6. Remove v from G and all adjacent edges to v. The remaining subgraph has n-1 vertices and by the induction hypothesis it can be properly colored by 6 colors. Since v has at most 5 adjacent vertices in G, then v can be colored with a color different from all of its neighbours. This ends the proof.

# Graph Colorings - Applications

## Scheduling exams

- many exams, each student have specified which exams he/she has to take
- how many different exam slots are needed? (a student cannot be at two different exams at the same time)

*Vertices:* courses

*Edges:* if there is a student taking both courses

*Exam slots:* colors

## Frequency assignments

- TV channels, mobile networks

# §10.1: Introduction to Trees

- A *tree* is a connected undirected graph that contains no circuits.
  - **Theorem:** There is a unique simple path between any two of its nodes.
- A (not-necessarily-connected) undirected graph without simple circuits is called a *forest*.
  - You can think of it as a set of trees having disjoint sets of nodes.
- A *leaf* node in a tree or forest is any pendant or isolated vertex. An *internal* node is any non-leaf vertex (thus it has degree ≥).

# Trees as Models

- Can use trees to model the following:
  - Saturated hydrocarbons
  - Organizational structures
  - Computer file systems
- In each case, would you use a rooted or a non-rooted tree?

# Some Tree Theorems

- Any tree with $n$ nodes has $e = n-1$ edges.
  - **Proof:** Consider removing leaves.
- A full $m$-ary tree with $i$ internal nodes has $n=mi+1$ nodes, and $\ell=(m-1)i+1$ leaves.
  - **Proof:** There are $mi$ children of internal nodes, plus the root. And, $\ell = n-i = (m-1)i+1$. □
- Thus, when $m$ is known and the tree is full, we can compute all four of the values $e$, $i$, $n$, and $\ell$, given any one of them.

# Some More Tree Theorems

- **Definition:** The *level* of a node is the length of the simple path from the root to the node.
  - The *height* of a tree is maximum node level.
  - A rooted *m*-ary tree with height *h* is called *balanced* if all leaves are at levels $h$ or $h-1$.
- **Theorem:** There are at most $m^h$ leaves in an *m*-ary tree of height *h*.
  - **Corollary:** An *m*-ary tree with $\ell$ leaves has height $h \geq \lceil \log_m \ell \rceil$ . If *m* is full and balanced then $h = \lceil \log_m \ell \rceil$.

# §10.2: Applications of Trees

- Binary search trees
  - A simple data structure for sorted lists
- Decision trees
  - Minimum comparisons in sorting algorithms
- Prefix codes
  - Huffman coding
- Game trees

# §10.3: Tree Traversal

- Universal address systems
- Traversal algorithms
  - Depth-first traversal:
    - Preorder traversal
    - Inorder traversal
    - Postorder traversal
  - Breadth-first traversal
- Infix/prefix/postfix notation