Discrete Mathematics Notes

Stasya

Fall 2024

Contents

1 Logic and Proofs **2**
1.1 Propositional Logic . . . 2
1.2 Propositional Equivalences . . . 3
1.3 Predicates and Quantifiers . . . 4
1.4 Nested Quantifiers . . . 5
1.5 Rules of Inference . . . 5
1.6 Introduction to Proofs . . . 7
1.7 Proof Methods and Strategy . . . 7

2 Sets, Functions, Sequences, Sums, and Matrices **9**
2.1 Sets . . . 9
2.2 Set Operations . . . 10
2.3 Functions . . . 10
2.4 Sequences and Summations . . . 11

3 Algorithms **13**

4 Number Theory and Cryptography **14**
4.1 Divisibility and Modular Arithmetic . . . 14
4.2 Integer Representation and Algorithms . . . 15
4.3 Primes and Greatest Common Divisors . . . 17
4.4 Solving Congruences . . . 19

5 Induction and Recursion **21**
5.1 Mathematical Induction . . . 21
5.2 Strong Induction and Well-Ordering . . . 21
5.3 Recursive Definitions and Structural Induction . . . 22
5.4 Recursive Algorithms . . . 24

6 Counting **26**
6.1 The Basics of Counting . . . 26
6.2 The Pigeonhole Principle . . . 26
6.3 Permuations and Combinations . . . 26
6.4 Binomial Coefficients and Identities . . . 26
6.5 Generalized Permuations and Combinations . . . 26

7 Discrete Probability **27**

8 Advanced Counting Techniques **28**
8.1 Applications of Recurrence Relations . . . 28
8.2 Inclusion-Exclusion . . . 28

9 Relations **29**

10 Graphs **30**
10.1 Graphs and Graph Models . . . 30
10.2 Graph Terminology and Special Types of Graphs . . . 30

11 Trees **31**

1 Logic and Proofs

1.1 Propositional Logic

A proposition is a declarative statement that is either true or false, but not both. We use letters to denote propositional variables. The conventional letters for propositional variables are p, q, r, s. True is notated as T and false is notated as F. An atomic proposition is a proposition that cannot be expressed in terms of simpler propositions.

Definition

Let p be a proposition. The negation of p, denoted by $\neg p$ (or $\bar{p}$) means "It is not the case that p".

The proposition $\neg p$ is read as "not p". The truth value of the negation of p, $\neg p$ is the opposite of the truth value of p.

The negation of a proposition can also be considered the result of the operation of the negation operator on a proposition.

The next operator is a connective and it is used to form new propositions from two or more existing propositions.

Definition

Let p and q be propositions. The conjunction of p and q, denoted by $p \wedge q$, is the proposition of "p and q". The conjunction $p \wedge q$ is true when both are true, and false when both are false.

Definition

Let p and q be propositions. The disjunction of p and q, denoted by $p \vee q$, is the proposition "p or q". The disjunction $p \vee q$ is false when both p and q are false, otherwise it is true.

Definition

Let p and q be propositions. The exclusive or of p and q, denoted by $p \oplus q$ is the proposition that is true when exactly one of p and q is true and is false otherwise.

There are other ways propositions can be combined.

Definition

Let p and q be propositions. The conditional statement $p \rightarrow q$, is the proposition, "if p, then q". The conditional statement $p \rightarrow q$ is false when p is true and q is false, and true otherwise. p is called the hypothesis and q is called the conclusion.

A conditional statement is also called an implication.

With $p \rightarrow q$, we can form three related conditional statements.

The first is the proposition $q \rightarrow p$, which is the converse of $p \rightarrow q$.

The contrapositive of $p \rightarrow q$ is $\neg q \rightarrow \neg p$.

The inverse of $p \rightarrow q$ is $\neg p \rightarrow \neg q$.

The contrapositive of a conditional statement is equal to it. We all two compound propositions equivalent when they always have the same truth values. The converse and inverse of a conditional statement are equivalent as well.

There is another way to combine propositions that expresses that two propositions have the same truth value.

Definition

Let p and q be propositions. The biconditional statement $p \leftrightarrow q$ is the proposition "p if and only q". This statement is true if p and q have the same truth values, and is false otherwise.

The negation operator is applied before all logical operators. Another general rule is that the conjunction operator takes precendence over the disjunction operator. It is an accepted rule that conditional and biconditional operators have lower precendence than the conjuction and disjunction operators.

A bit is a symbol with two values, 0 and 1. 1 is true, and 0 is false.

1.2 Propositional Equivalences

Definition

A compound proposition that is always true, no matter what the truth values of the propositional values that occur in it, is called a tautology. A compound proposition that is always false is called a contradiction. If it is neither a tautology or a contradiction, it is called a contingency.

Compound propositions that have the same truth values in all possible cases are called logically equivalent.

Definition

The compound propositions p and q are called logically equivalent if $p \leftrightarrow q$ is a tautology. The notation $p \equiv q$ denotes that p and q are logically equivalent.

We can establish logical equivalence of more than two compound propositions. Generally 2^n rows are required if a compound proposition involves n propositional variables.

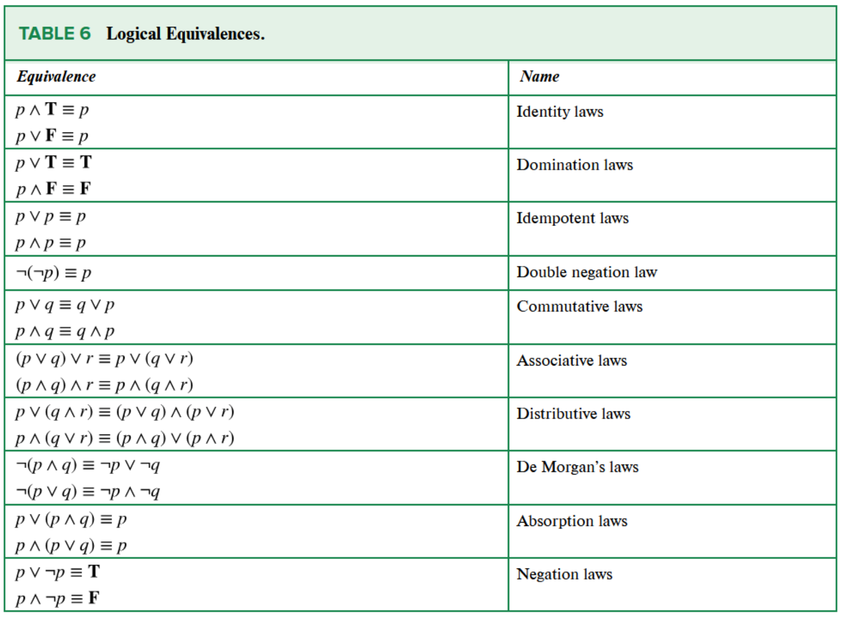

TABLE 6 Logical Equivalences.

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

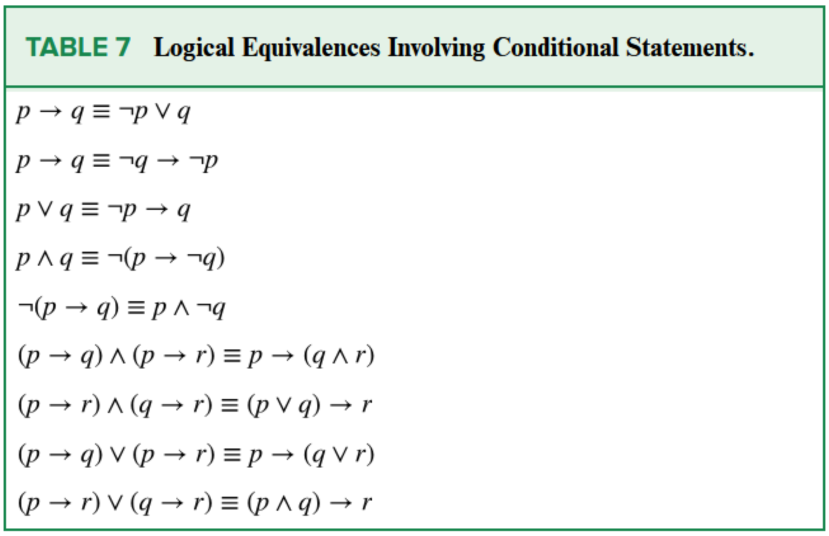

TABLE 7 **Logical Equivalences Involving Conditional Statements.**
$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

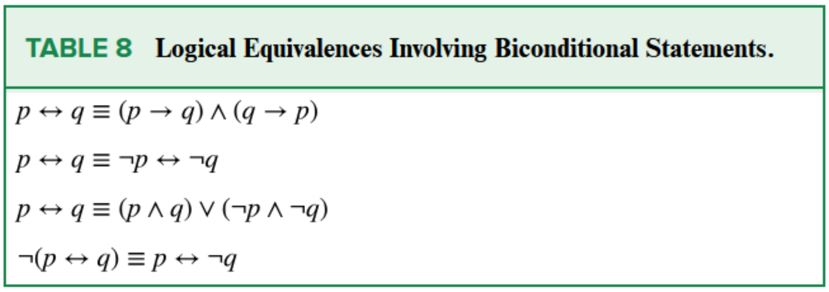

TABLE 8 **Logical Equivalences Involving Biconditional Statements.**
$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Credit: Rosen's Discrete Mathematics 8e

When using De Morgan's Laws, remember to change the logical connective after you negate.

De Morgan's Laws tell us how to negate conjunctions and how to negate disjunctions. The equivalence $\neg(p \vee q) \equiv \neg p \wedge \neg q$ tells us that the negation of a disjunction is formed from the conjunction of the negations of the component propositions. The negation of a conjunction also gives you the disjunction of the negations of component propositions.

A compound proposition is satisfiable if there is an assignment of truth values to its variables that make it true. If no such assignments exist, then it is unsatisfiable.

1.3 Predicates and Quantifiers

We will introduce predicate logic now. For example in the statement x is greater than 3, the variable is x and the predicate is "is greater than 3".

In general a statement involving n variables $x_1, x_2, \cdots, x_n$ can be denoted as $P(x_1, x_2, \cdots, x_n)$.

Definition

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

"$P(x)$ for all values of x in the domain."

The notation $\forall x P(x)$ denotes the universal quantification of $P(x)$. Here $\forall$ is called the universal quantifier. An element for which $P(x)$ is false is called a counterexample to $\forall x P(x)$.

Definition

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

"There exists an element in x in the domain such that $P(x)$"

We use the notation $\exists x P(x)$ for the existential quantification of $P(x)$.

Remember the truth value for both $\forall x P(x)$ and $\exists x P(x)$ depends on the domain.

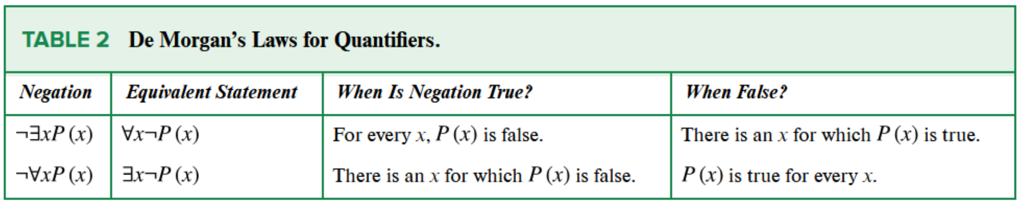

TABLE 2 De Morgan's Laws for Quantifiers.

Negation	*Equivalent Statement*	*When Is Negation True?*	*When False?*
$\neg\exists x P(x)$	$\forall x \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg\forall x P(x)$	$\exists x \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

Credit: Rosen's Discrete Mathematics 8e

1.4 Nested Quantifiers

A nested quantifier is when one quantifier is in the scope of another.

Here is table that summarizes different quantifications involving two variables.

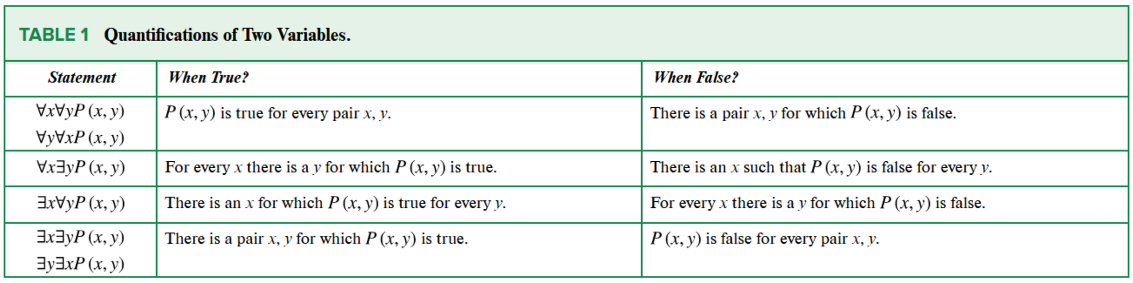

TABLE 1 Quantifications of Two Variables.

Statement	*When True?*	*When False?*
$\forall x \forall y P(x, y)$ $\forall y \forall x P(x, y)$	$P(x, y)$ is true for every pair x, y.	There is a pair x, y for which $P(x, y)$ is false.
$\forall x \exists y P(x, y)$	For every x there is a y for which $P(x, y)$ is true.	There is an x such that $P(x, y)$ is false for every y.
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for every y.	For every x there is a y for which $P(x, y)$ is false.
$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	There is a pair x, y for which $P(x, y)$ is true.	$P(x, y)$ is false for every pair x, y.

1.5 Rules of Inference

Rules of inference are our basic tools for establishing the truth of statements.

An argument in propositional logic is a sequence of propositions. All but the final proposition in the argument are called premises and the final proposition is called the conclusion. An argument is valid if the truth of all its premises implies 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 what particular propositions are substituted for the propositional variables in the premises, the conclusion is true if the premises are all true.

The tautology $(p \wedge (p \to q)) \to q$ is the basis of the rule of inference modus ponens, or the law of detachment, basically: if p and $p \to q \therefore q$.

Yet again another chart I took from Rosen.

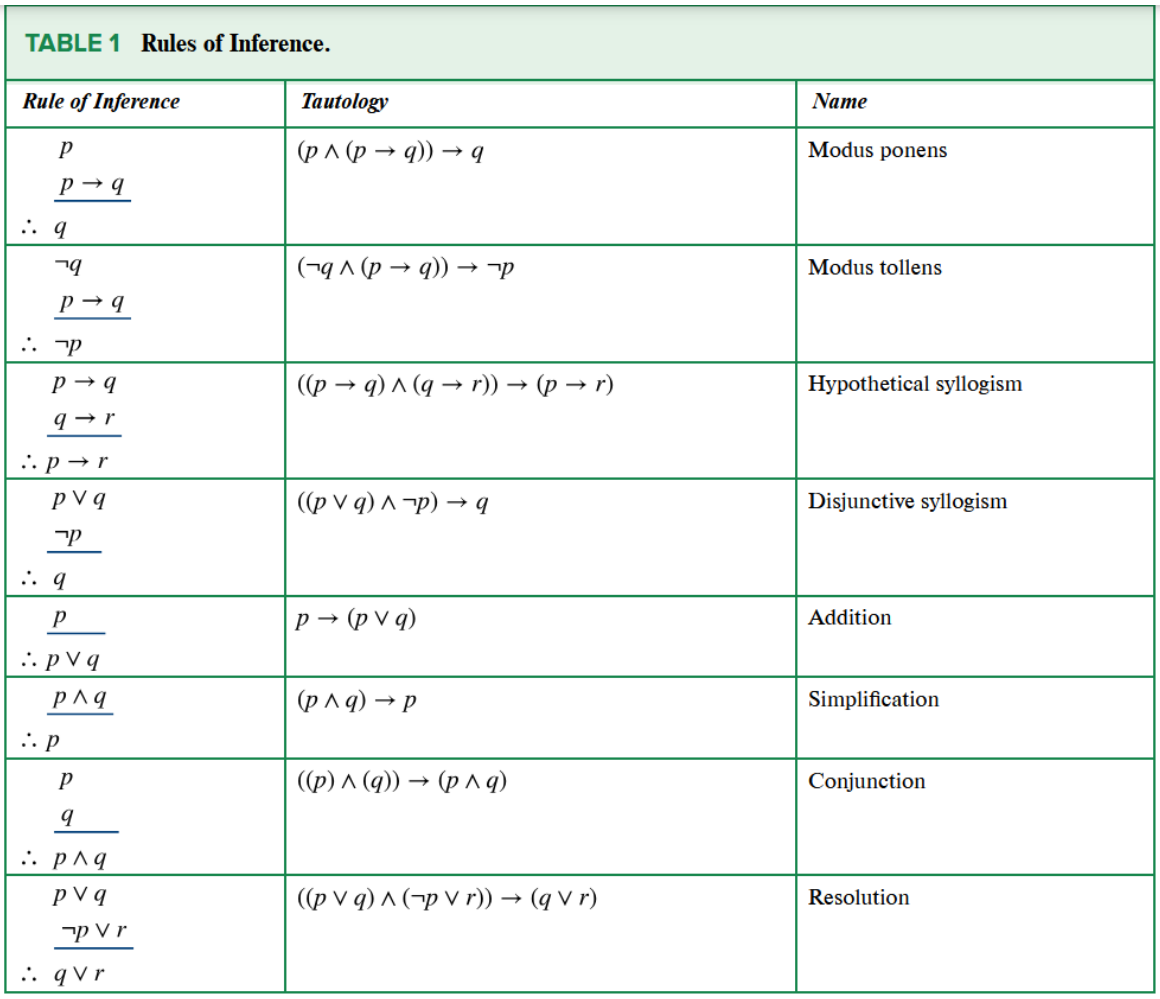

TABLE 1 **Rules of Inference.**

Rule of Inference	*Tautology*	*Name*
p $\underline{p \to q}$ $\therefore q$	$(p \land (p \to q)) \to q$	Modus ponens
$\neg q$ $\underline{p \to q}$ $\therefore \neg p$	$(\neg q \land (p \to q)) \to \neg p$	Modus tollens
$p \to q$ $\underline{q \to r}$ $\therefore p \to r$	$((p \to q) \land (q \to r)) \to (p \to r)$	Hypothetical syllogism
$p \lor q$ $\underline{\neg p}$ $\therefore q$	$((p \lor q) \land \neg p) \to q$	Disjunctive syllogism
$\underline{p}$ $\therefore p \lor q$	$p \to (p \lor q)$	Addition
$\underline{p \land q}$ $\therefore p$	$(p \land q) \to p$	Simplification
p $\underline{q}$ $\therefore p \land q$	$((p) \land (q)) \to (p \land q)$	Conjunction
$p \lor q$ $\underline{\neg p \lor r}$ $\therefore q \lor r$	$((p \lor q) \land (\neg p \lor r)) \to (q \lor r)$	Resolution

Fallacies resemble rules of inference, the difference lies in that they are based in contingencies rather than tautologies.

There are also some rules of inference for statements involving quantifiers. This is what Rosen summarized:

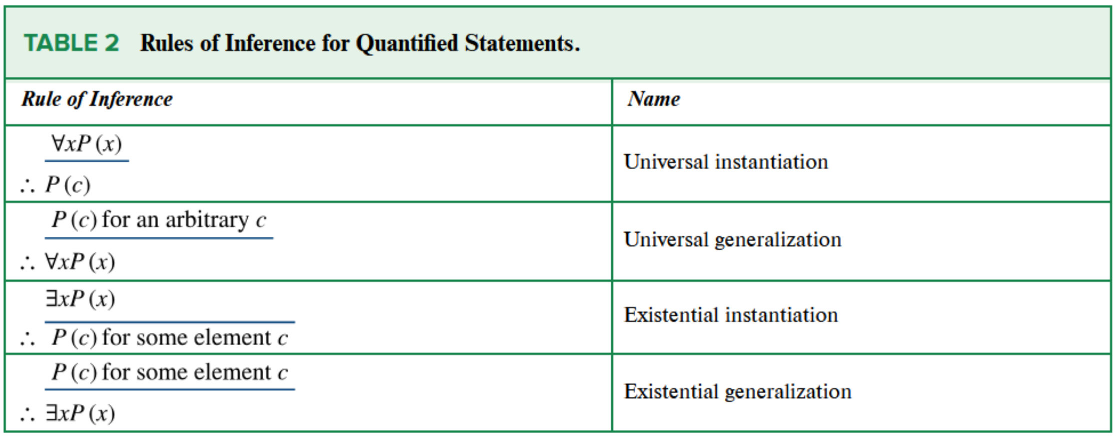

TABLE 2 **Rules of Inference for Quantified Statements.**

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

Because universal instantiation and modus ponens are so often used together, the combination of rules is called universal modus ponens. This rule tells us if $\forall x(P(x) \to Q(x))$ is true, and if $P(a)$ is true for a particular element a in the domain of the universal quantifier, then $Q(a)$ must also be true. Note that by universal instantiation $P(a) \to Q(a)$ is true. By modus ponens, $Q(a)$ must also be true.

This is how it is described:

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

There is also universal modus tollens:

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

1.6 Introduction to Proofs

A proof is a valid argument that establishes the truth of a mathematical statement.

A theorem is a statement that can be shown to be true. We show the truth using a proof.

A less important theorem that is helpful in the proof of other results is called a lemma.

A corollary is a theorem that can be established directly from a theorem that has been proved.

A conjecture is a statement that is being proposed to be a true statement.

A direct proof of a conditional statement $p \rightarrow q$ is constructed with the assumption that p is true with the goal for the combination of p being true and q being false to never happen.

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 $n = 2k + 1$. Two integers have the same parity when both are even or both are odd; they have opposite parity when one is even and the other is odd.

An indirect proof is a type of proof that is not direct. One example is proof by contraposition. P roofs by contraposition use the fact that $p \rightarrow q$ is equal to $\neg q \rightarrow \neg p$. We have to show $\neg q \rightarrow \neg p$ is true to show that $p \rightarrow q$ is true as well.

We can prove that $p \rightarrow q$ is true if p is false as well. This is called a vacuous proof.

If we show that q is true in $p \rightarrow q$ this is called a trivial proof.

Definition

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

Suppose we want to prove a statement p is true. Suppose that we can find a contradiction q such that $\neg p \rightarrow q$ is true. Because q is false, but $\neg p \rightarrow q$ is true, we can conclude $\neg p$ is false, meaning p is true.

To find a contradiction q that might help us find that p is true, we can show that $\neg p \rightarrow (r \wedge \neg r)$ is true for some proposition r. This is called a proof by contradiction.

To prove a theorem in the form $p \leftrightarrow q$, we show that both $p \rightarrow q$ and $q \rightarrow p$ are both true.

Many incorrect arguments are based on a fallacy called begging the question or circular reasoning. This fallacy occurs when one or more steps of a proof are based on the truth of the statement being proved.

1.7 Proof Methods and Strategy

Suppose we have a conditional $(p_1 \vee p_2 \vee \cdots \vee p_n) \rightarrow q$. We can separate the proof into different cases, called proof by cases by proving each of the n conditional statements.

Some theorems can be proved by examining a relatively small number of examples. This is called proof by exhaustion.

The phrase "without loss of generality" (WLOG) means that we assert by proving one case of a theorem, no additional argument is required to prove other specified cases.

Many theorems are assertions that objects of a particular type exist. A theorem of this type is a proposition in the form $\exists x P(x)$, where P is a predicate. A proof of a proposition in this form is called an existence proof. Sometimes an existence proof can be given by finding an element a, called a witness, such that $P(a)$ is true. This existence proof is called constructive. If we prove $\exists x P(x)$ is true in another way it can be nonconstructive.

For a uniqueness proof, we need two parts:

- Existence: We show that an element x with the desired property exists.
- Uniqueness: We show that if x and y both have the desired property, $x = y$

Forward reasoning is a proof when you start with your premises. You construct a proof with steps leading to the conclusion. It may sometimes be helpful to use backwards reasoning, we find a statement p that we can prove $p \to q$.

2 Sets, Functions, Sequences, Sums, and Matrices

2.1 Sets

Sets are used to group objects together.

A set is an unordered collection of objects, called elements or members of the set. A set contains its elements. We write $a \in A$ to denote a is in an element of set A. The notation $a \notin A$ denotes a is not an element of set A.

Here are some sets to remember:

- $\mathbb{N}$ is the set of all natural numbers
- $\mathbb{Z}$ is the set of all integers
- $\mathbb{Z}^+$ is the set of all positive integers
- $\mathbb{Q}$ is the set of all rational numbers
- $\mathbb{R}$ is the set of all real numbers
- $\mathbb{R}^+$ is the set all positive real numbers
- $\mathbb{C}$ is the set of all complex numbers

Two sets are equal only if they contain the same elements.

An empty set is notated as $\emptyset$.

A set with one element is a singleton set.

Set A is a subset of set B and set B is the superset of set A if every element of A is also an element of B. To indicate A is a subset of B we write $A \subseteq B$. For the equivalent superset, we write $B \supseteq A$.

For every nonempty set S, there is a guarantee to have at least two subsets, the empty set and the set S itself.

When we want to say that A is a subset of B, but $A \neq B$, we can write $A \subset B$.

If there are n distinct elements in a set S, then the set is finite and n is the cardinality of S. The cardinality of S is denoted as $|S|$.

Otherwise, the set is infinite if it is not finite.

Definition

Given a set S, the power set of S is the set of all subsets of the set S. The power set of S is defined as $\mathcal{P}(S)$.

The power set of a set has 2^n elements. Because sets are unordered, we need to represent ordered collections using ordered n-tuples.

Definition

The ordered n-tuple $(a_1, a_2, \cdots, a_n)$ is the ordered collection that has a_1 as its first element,

2.2 Set Operations

If we let A and B be sets, the union of the sets, $A \cup B$, is the set that contains those elements that are either in A or B, or in both.

The intersection of the sets, $A \cap B$, is the set containing those elements in both A and B.

Two sets are called disjoint if the intersection of the sets is an empty set.

The difference of sets A and B, or $A - B$ is the set containing those elements that are in A but not in B. It is also called the complement of B with respect to A.

Definition

Let U be the universal set. The complement of set A denoted as $\overline{A}$ is the complement of A with respect to U. Therefore the complement of the set A is $U - A$.

Much like the last chapter, there are some set identities and properties

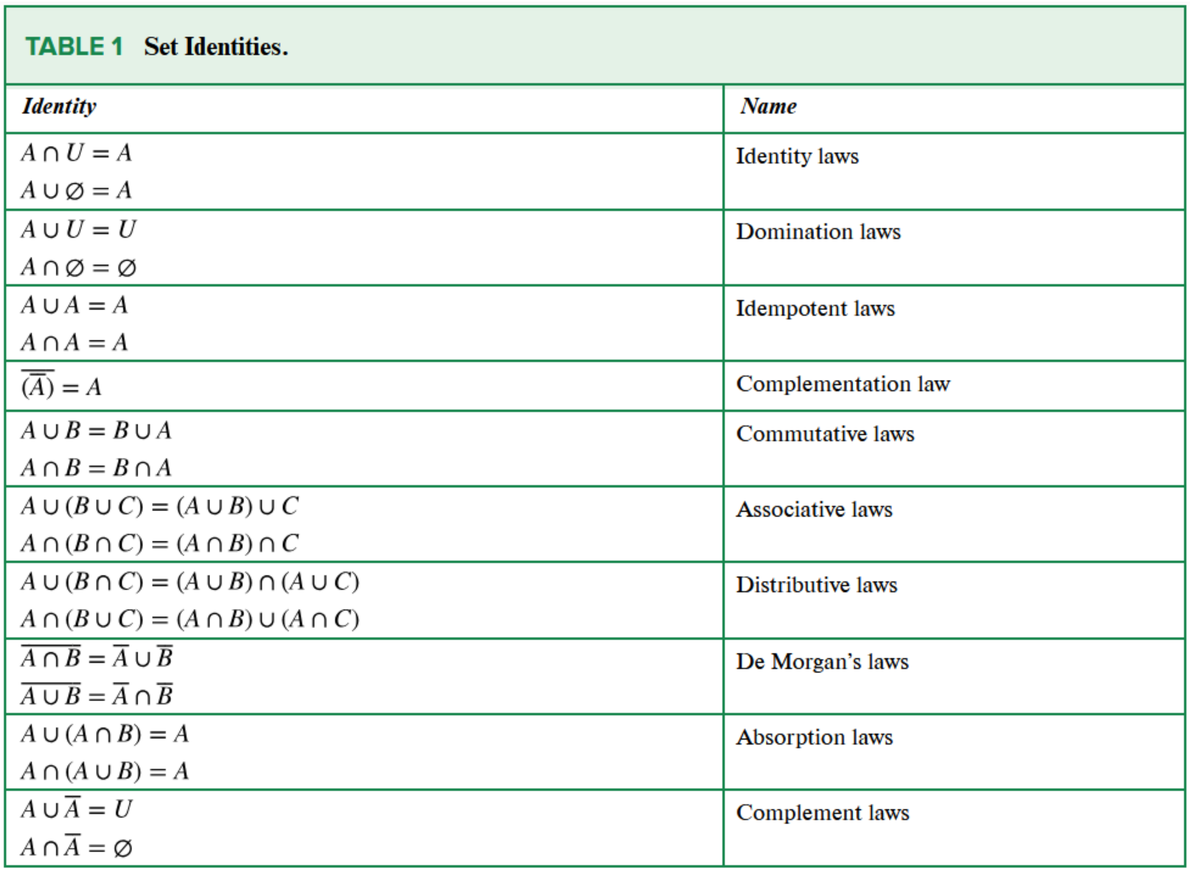

TABLE 1 Set Identities.

Identity	*Name*
$A \cap U = A$ $A \cup \emptyset = A$	Identity laws
$A \cup U = U$ $A \cap \emptyset = \emptyset$	Domination laws
$A \cup A = A$ $A \cap A = A$	Idempotent laws
$\overline{(\overline{A})} = A$	Complementation law
$A \cup B = B \cup A$ $A \cap B = B \cap A$	Commutative laws
$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$	Associative laws
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive laws
$\overline{A \cap B} = \overline{A} \cup \overline{B}$ $\overline{A \cup B} = \overline{A} \cap \overline{B}$	De Morgan's laws
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	Absorption laws
$A \cup \overline{A} = U$ $A \cap \overline{A} = \emptyset$	Complement laws

Credits to Rosen again.

The union of a collection of sets is the set that contains those elements that are members of at least one set in the collection.

The intersection of a collection of sets is the set that contains those elements that are members of all sets in the collection.

2.3 Functions

Definition

Let A and B be empty sets. A function f from A to B is an assignment of exactly one element of B to each element of A. We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A. If f is a function from A to B, we write $f : A \to B$.

If f is a function from A to B, we say A is the domain of f and B is the codomain of f. Also if $f(a) = b$, we can say b is the image of a and a is a preimage of b. The range, or image, is the set of all images of elements of A. Also, if f is a function from A to B, we say that f maps from A to B.

A function is called real-valued if its codomain is the set of real numbers and integer-valued if the codomain is the set of integers.

Some functions never assign the same value to two different domain elements. These are called one-to-one functions, or injective functions.

A function is called surjective or onto when the range and codomain are equal.

If a function is both surjective and injective, then it is bijective.

Only a one-to-one function can be invertible because the inverse of a one-to-one function exists.

2.4 Sequences and Summations

Sequences are ordered lists of elements. The terms of a sequence can be specified by providing a formula for each term of the sequence.

A sequence is used to represent an ordered list. We use the notation a_n to denote the image of the integer n. We call a_n a term of the sequence.

A geometric progression is a sequence in the following form:

$$a, ar, ar^2, \cdots, ar^n, \cdots$$

where the initian term a and common ratio r are real numbers.

An arithmetic progression is a seuqnece in the form:

$$a, a + d, a + 2d, \cdots, a + nd, \cdots$$

where the initial term a and the common difference d are real numbers.

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 in the sequence.

A sequence is called a solution of a recurrence relation if its terms satisfy the recurrence relation.

Definition

The Fibonacci sequence, $f_0, f_1, f_2, \cdots$, is defined by the initial conditions $f_0 = 0, f_1 = 1$, and the recurrence relation:

$$f_n = f_{n-1} + f_{n-2}$$

for $n = 2, 3, 4, \cdots$.

Now we introduce the summation notation.

We use the notation $\sum_{j=m}^{n} a_j$ to represent $a_m + a_{m+1} + \cdots + a_n$.

Here j is the index of summation and is abitrary.

Sums of terms of geometric progressions commonly arise.

$$\sum_{j=0}^{n} ar^j = \frac{ar^{n+1} - a}{r - 1}$$

when $r \neq 1$ and $(n + 1)a$ when $r = 1$.

Here is some formulae for commonly occurring sums:

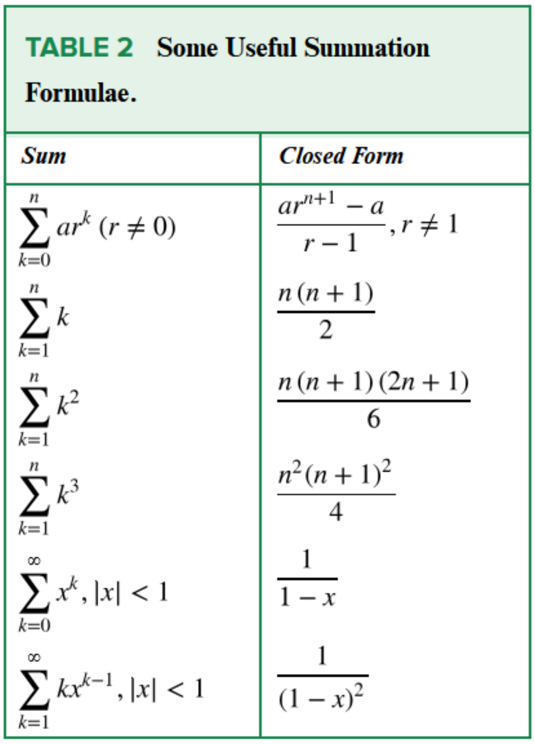

TABLE 2 Some Useful Summation Formulae.

Sum	*Closed Form*
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1}-a}{r-1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x \rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1}, \lvert x \rvert < 1$	$\frac{1}{(1-x)^2}$

Credits to Rosen again.

3 Algorithms

An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.

There are many properties of algorithms to keep in mind:

- Input - input values from a specified set
- Output - from each set of input values an algorithm produces output values
- Definiteness - the steps of an algorithm must be defined precisely
- Correctness - an algorithm should produce correct output values for each set of input values
- Finiteness - an algorithm should produce the desired output after a finite number of steps for any input
- Effectiveness - it must be possible to perform each step of an algorithm exactly and in a finite amount of time
- Generality - the procedure should be applicable for all problems of the desired form

The first algorithm to present is the linear search, or sequential search. This search begins by comparing a x and a_1 and continues with each a_n until a match is found.

The binary search works when the list is sorted. We first split the list into two smaller sublists of the same size, and keep splitting it up based on the comparison to the term to be found the middle term.

The bubble sort is one of the simplest sorting algorithms. It puts a list into increasing order by successively comparing adjacent elements.

The insertion sort begins with the second element and compares it with the the first element. Then the third element is compared with the first and then the second if it is larger than the first.

Many algorithms are designed to solve optimization problems. This means they want to maximize or minimize some parameter. Algorithms that make what seems to be the "best" choice at each step are called greedy algorithms.

An example would be the cashier's algorithm which makes changes using the fewest coins possible when change is made from quarters, dimes, nickels, and pennies.

The halting problem is a interesting problem. It asks whether there is a procedure that can input a computer program and determine whether the program will eventually stop.

The reason there isn't one is because you do not know if it will never halt or you haven't waited long enough for it to terminate.

4 Number Theory and Cryptography

4.1 Divisibility and Modular Arithmetic

Definition

If a and b are integers with $a \neq 0$, we say that a divides b if there is an integer c such that $b = ac$ (or equivalently, if $\frac{b}{a}$ is an integer). When a divides b we say that a is a factor or divisor of b, and that b is a multiple of a. The notation $a \mid b$ denotes that a divides b. We write $a \nmid b$ when a does not divide b.

We can express $a \mid b$ using quantifiers as $\exists c(ac = b)$.

There are some basic properties of divisibility of integers:

- If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$
- If $a \mid b$, then $a \mid bc$ for all integers c
- If $a \mid b$ and $b \mid c$, then $a \mid c$

Corollary 4.1

If a, b, and c are integers, where $a \neq 0$, such that $a \mid b$ and $a \mid c$, then $a \mid mb + nc$ whenever m and n are integers.

When an integer is divided by a positive integer, there is a quotient and a remainder, as the division algorithm shows.

Theorem 4.2: 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 $a = dq + r$.

In the equality given in the division algorithm, d is called the divisor, a is called the divided, q is called the quotient, and r is called the remainder. The notation $q = a$ **div** d is used to denote quotient, and $r = a \mod d$ is used to notate the remainder.

When a is an integer and d is a positive integer, a **div** $d = \lfloor a/d \rfloor$ and $a \mod d = a - d$.

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 $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m. We say that $a \equiv b \pmod{m}$ is a congruence and that m is its modulus. If a and b are not congruent modulo m, we write $a \not\equiv b \pmod{m}$.

Theorem 4.3

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

Theorem 4.4

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 $a = b + km$.

Theorem 4.5

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

$$ac \equiv bd \pmod{m}$$

Corollary 4.6

Let m be a positive integer and let a and b be integers. Then:

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

and

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

We can define arithmetic operations on Z_m, the set of nonnegative integers less than m, that is, the set $0, 1, \ldots, m-1$. We definte the the addition of these integers as:

$$a +_m b = (a+b) \mod m$$

We can define the multiplication of these integers as

$$a \cdot_m b = (a \cdot b) \mod m$$

Here are some properties of these:

- Closure - If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to z_m.
- Associativity - If a, b, and c, belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m) c$.
- Commutativity - If a and b belong to Z_m, then $a +_m b = b +_m a$ and $a \cdot_m b = b \cdot_m a$.
- Identity elements - The elements 0 and 1 are identity elements for addition and multiplication modulo m, respectively.
- Additive Inverses - If $a \neq 0$ belongs to Z_m, then $m - a$ is an additive inverse of a modulo m and 0 is its own additive inverse. That is, $a +_m (m - a) = 0$ and $0 +_m 0 = 0$.
- Distributivity - If a, b, and c belong to Z_m, then $a \cdot_m (b +_m c) = (a \cdot_m n) +_m (a \cdot_m c)$ and $(a +_m b) \cdot_m c = (a \cdot_m c) +_m (b \cdot_m c)$

4.2 Integer Representation and Algorithms

Theorem 4.7

Let b be an integer greater than 1. Then if n is a positive integer, it can be expressed uniquely in the form:

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

where k is a nonnegative integer, a_0, a_1,..., a_k are nonnegative integers less than b, and $a_k \neq 0$.

Choosing 2 as the base gives binary expansions of integers. In binary notation each digit is either a 0 or a 1. In other words, the binary expansion of an integer is a just a bit string.

Base 8 expansions are called octal expansions and base 16 expansions are hexadecimal expansions.

We can describe an algorithm for constructing the base b expansion of an integer n.

First divide n by b to obtain a quotient and remainder:

$$n = bq_0 + a_0$$

assuming a_0 is positive and less than b.

The remainder, a_0, is the rightmost digit in the base b expansion of n. Next, divide q_0 by b to obtain:

$$q_0 = bq_1 + a_1 \qquad 0 \leq a_1 < b$$

Continue this process, successively dividing the quotients by b, obtaining additional base b digits as the remainders.

The algorithms for performing operations with integers using their binary expansions is important.

Throughout the following, lets suppose that the binary expansions of a and b are:

$$a = (a_{n-1}a_{n-2}\ldots a_1a_0)_2, b = (b_{n-1}b_{n-2}\cdots b_1b_0)_2$$

so that a and b each have n bits.

Let's first consider the adding two integers in binary notation. The procedure is as follows:

First add their rightmost bits:

$$a_0 + b_0 = c_0 \cdot 2 + s_0$$

where s_0 is the rightmost bit in the binary expansion of $a + b$ and c_0 is the carry, either a 0 or 1.

Continue this:

$$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$$

This procedure produces the binary expansion of the sum:

$$a + b = (s_ns_{n-1}s_{n-2}\ldots s_1s_0)_2$$

Now we consider multiplication. Using the distributive law it is easy to see that

$$\begin{aligned} ab &= a(b_02^0 + b_12^1 + \cdots + b_{n-1}2^{n-1}) \\ &= a(b_02^0) + a(b_12^1) + \cdots + a(b_{n-1}2^{n-1}) \end{aligned}$$

There is an algorithm for div and mod as well. Given integers a and d, where $d > 0$, we can find $q = a\textbf{div}d$ and $r = a \mod d$ using the algorithm below.

We can show that this algorithm uses $O(q \log a)$ bit operations when $a > d$.

Given a as an integer, and d as a positive integer, we can make q equal to 0 and $r = |a|$.

While $r \geq d$, r gets assigned to $r - d$ and q gets assigned $q + 1$.

If $a < 0$ and $r > 0$ then r gets assigned $d - r$ and q gets assigned $-(q + 1)$.

We return the quotient as q and the remainder as r.

When a is divided by b, we need $O(n^2)$ bit operations to find the quotient and remainder.

In cryptography it is important to be able to find $b^n \mod m$ efficiencly without using an excessive amount of memory.

First observe that we can avoid using a large amount of memory if we compute $b^n \mod m$ by successively computing $b^k \mod m$ for $k = 1, 2, \ldots, n$ using that fact that $b^{k+1} \mod m = b(b^k \mod m) \mod m$.

To motivate the fast modular exponentiation algorithm, we can first explain how to use the binary expansion of n.

Say that $n = (a_{k-1} \dots a_1 a_0)_2$ to compute b^n. Firstly, note that:

$$b^n = b^{a_{k+1} \cdot 2^{k-1} + \dots + a_1 \cdot 2 + a_0} = b^{a_{k=1} \cdot 2^{k-1}} \dots b^{a_1 \cdot 2} \cdot b^{a_0}$$

This shows that to compute b^n we only have to compute b, b^2, b^4, $\dots$, b^{2^k}.

Note that $(b^{2^n})^2 = b^{2^{n+1}}$ when n is a nonnegative integer.

The algorithm successively finds $b \mod m, b^2 \mod m, b^4 \mod m, dots, b^{2^{k-1}} \mod m$ and multiplies together those terms $b^{2^i} \mod m$ where $a_j = 1$, finding the remainder of the product when divided by m after each multiplication. We only need to perform $O(\log_2(n))$ multiplications.

We can also show the most efficient algorithm as seen:

Let b be an integer and $n = (a_{k-1} a_{k-2} \dots a_1 a_0)_2$ and m be positive integers.

Let x be 1 and power be $b \mod m$.

For all values 0 to $k-1$, if $a_i = 1$, then x becomes $(x \cdot \text{power}) \mod m$ and power becomes power·power $\mod m$.

The return value for this algorithm is x, which is $b^n \mod m$.

This algorithm only uses $O((\log m)^2 \log n)$ bit operations to find $b^n \mod m$.

4.3 Primes and Greatest Common Divisors

Positive integers that have exactly two different positive integer factors are called prime.

Definition

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

Remember that 1 is not prime, and that an integer n is composite if and only if there exists an integer a such that $a \mid n$ and $1 < a < n$.

Theorem 4.8: The Fundamental Theorem of Arithmetic

Every 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.

Theorem 4.9

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

From this theorem above, it follows that an integer is prime if it is not divisible by any prime less than or equal to its square root. This leads to the brute-force algorithm known as trial division. To use trial division, we divide n by all primes not exceeding $\sqrt{n}$ and conclude that n is prime if it is not divisible by any of these primes.

Because every integer has a prime factorization, it would be useful to have a procedure for finding the prime factorization. Start with the theorem above, and find if there is a prime factor not exceeding $\sqrt{n}$. If there is none found, then continue by factoring n/p. Note that n/p will have no prime factors less than p. If n/q has a prime factor q, then continue by factoring $n/(pq)$. There are infinitely many primes.

Theorem 4.10: The Prime Number Theorem

The ratio of $\pi(x)$, the number of primes not exceeding x, and $x/\ln x$ approaches 1 as x grows without bound.

Every odd integer is in one of the two arithmetic progressions $4k+1$ or $4k+3$, where $k = 1, 2, \cdots$.

The largest integer that divides two integers is called the greatest common divisor of these integers.

Definition

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

Definition

The integers a and b are relatively prime if their greatest common divisor is 1.

Definition

THe integers a_1, a_2,..., a_n are pairwise relatively prime if $\gcd(a_i, a_k) = 1$ whenever $1 \leq i < j \leq n$.

Prime factorizations can be used to find the least common multiple of the integers.

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. The least common multiple of a and b is denoted by lcm(a, b).

Theorem 4.11

Let a and b be positive integers. Then

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

There is a more efficient way of finding the greatest common divisor using the Euclidean algorithm.

Lemma 4.12

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

Proof. If we can show that the common divisors of a and b are the same as the common divisors of b and r, we will have shown that $\gcd(a, b) = \gcd(b, r)$, because both pairs must have the same greatest common divisors.

So suppose that d divides both a and b. Then it follows that d also divides $a - bq = r$. Hence, any common divisor of a and b is also a common divisor of b and r.

Likewise, suppose that d divides both b and r. Then d also divides $bq + r = a$. Hence, any common divisor of b and r is also a common divisor of a and b.

Consequently, $\gcd(a, b) = \gcd(b, r)$. □

An important result is that the greatest common divisor of two integers a and b can be expressed in the form:

$$sa + tb$$

where s and t are integers. In other words, $\gcd(a, b)$ can be expressed as a linear combination with integer coefficients of a and b.

Theorem 4.13: Bezout's Theorem

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

Lemma 4.14

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

Lemma 4.15

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

Theorem 4.16

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

4.4 Solving Congruences

A congruence of the form

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

where m is a positive integer, a and b are integers, and x is a variable, is called a linear congruence.

Our goal is to solve th linear congruence $ax \equiv b \pmod{m}$.

One method is to use an integer $\overline{a}$ such that $\overline{a}a \equiv 1 \pmod{m}$, if such an integer exists. Such an integer $\overline{a}$ is to be an inverse of a modulo m.

Theorem 4.17

If a and m are relatively prime integers and $m > 1$, then an inverse of a modulo m exists. Furthermore, this inverse is unique modulo m. (That is there is a unique positive integer $\overline{a}$ less than m that is an inverse of a modulo m and every other inverse of a modulo m is congruent to $\overline{a}$ modulo m.)

We can design a more efficient algorithm than brute force to find an inverse of a modulo m when $\gcd(a, b) = 1$ using the steps of the Euclidean algorithm.

Once we have an inverse $\overline{a}$ of a modulo m, we can solve the congruence $ax \equiv b \pmod{m}$ by multiplying both sides of the linear congruence by $\overline{a}$.

The Chinese remainder theory states that when the moduli of a system of linear congruences are pairwise relatively prime, there is a unique solution of the system modulo of the product of the moduli.

Theorem 4.18: The Chinese Remainder Theorem

Let m_1, m_2,..., m_n be pairwise relatively prime positive integers greater than one and a_1, a_2,...,a_n arbitrary integers. Then the system

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

has a unique solution modulo $m = m_1m_2 \cdots m_n$. (That is, there is a solution x with $0 \le x < m$, and all other solutions are congruent modulo m to this solution.)

Suppose that m_1, m_2,. . . , m_n are pairwise relatively prime moduli and let m be their product. By the Chinese remainder theorem, we can show that an integer a with $0 \leq a < m$ can be uniquely represented by the n-tuple consisting of its remainders upon division by m_i, $i = 1, 2, \ldots, n$. That is, we can uniquely represent a by

$$(a \mod m_1, a \mod m_2, \ldots, a \mod m_n)$$

Theorem 4.19: Fermat's Little Theorem

If p is 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

$$a^p \equiv a \pmod p$$

Fermat's little theorem tells us that if $a \in \mathbb{Z}_p$, then $a^{p-1} = 1$ in $\mathbb{Z}_p$.

Definition

Let b be a positive integer. If n is a composite positive integer, and $b^{n-1} \equiv 1 \pmod n$, then n is called a pseudoprime to the base b.

Definition

A composite integer n that satisfies the congruence $b^{n-1} \equiv 1 \pmod n$ for all positive integers b with $\gcd(b, n) = 1$ is called a Carmichael number.

In the set of positive real numbers, if $b > 1$ and $x = b^y$, we say that y is the logarithm of x to the base b. Here, we will show that we can also define the concept of logarithms modulo p of positive integers, where p is a prime.

Definition

A primitive root modulo of prime p is an integer r in $\mathbb{Z}_p$ such that every nonzero element of $\mathbb{Z}_p$ is a power of r.

An important fact in number theory is that there is a primitive root modulo p for every prime p.

Suppose that p is prime and r is a primitive root modulo p. If a is an integer between 1 and $p - 1$, that is, a nonzero element of $\mathbb{Z}_p$, we know that there is a unique exponent e such that $r^e = a$ in $\mathbb{Z}_p$, that is, $r^e \mod p = a$.

Definition

Suppose that p is a prime, r is a primitive root modulo p, and a is an integer between 1 and $p - 1$ inclusive. If $r^e \mod p = a$ and $0 \leq e \leq p - 1$, we say that e is the discrete logarithm of a modulo p to the base r and we write $log_r a = e$ (where the prime p is understood).

5 Induction and Recursion

5.1 Mathematical Induction

To prove $P(n)$ is true for all integers n, where $P(n)$ is a propositional function, we complete two steps:

1. We verify that $P(1)$ is true.
2. We show that the conditional statement $P(k) \to P(k+1)$ is true for all positive integers k.

To complete the inductive step of a proof using the principle of mathematical induction, we assume that $P(k)$ is true for an arbitrary positive integer k and show that under this assumption, $P(k+1)$ must also be true. The assumption that $P(k)$ is true is called the inductive hypothesis. One we complete both steps in a proof by mathematical induction, we have shown that $P(n)$ is true for all positive integers n, that is, we have shown that $\forall n P(n)$ is true where the quantification is over the set of positive integers. In the inductive set, we show that $\forall k(P(k) \to P(k+1))$ is true, where again, the domain is set of all positive integers.

Written as a rule of inference this can be written as:

$$(P(1) \land \forall k(P(k) \to P(k+1))) \to \forall n P(n)$$

where the domain is the set of positive integers.

The first thing we do to prove that $P(n)$ is true for all positive integers n is to show that $P(1)$ is true. This amounts to showing that the particular statement obtained when n is replaced by 1 in $P(n)$ is true. Then we must show that $P(k) \to P(k+1)$ is true for every positive integer k. To prove that this conditional statement is true for every positive integer k, we need to show that $P(k+1)$ cannot be false when $P(k)$ is true. This can be accomplished by assuming $P(k)$ is true and showing that under this hypothesis $P(k+1)$ must also be true.

A guideline for proofs by mathematical induction:

1. Express the statement that is to be proved in the form "for all $n \geq b, P(n)$" for a fixed integer b. For statements of the form "$P(n)$ for all positive integers n", let $b = 1$, and for all statements of the form "$P(n)$ for all nonnegative integers n", let $b = 0$.
2. Show that $P(b)$ is true, taking care that the correct value of b is used.
3. Identify the inductive hypothesis, in the form "Assume $P(k)$ is true for an arbitrary fixed integer $k \geq b$."
4. State what needs to be proved under the assumption that the inductive hypothesis is true. That is, write out what $P(k+1)$ says.
5. Prove the statement $P(k+1)$ making use of the assumption $P(k)$.
6. Clearly identify the conclusion of the inductive step.
7. State the conclusion.

5.2 Strong Induction and Well-Ordering

The basis step of a proof by strong induction is the same as a proof of the same result using mathematical induction. That is, in a strong induction proof $P(n)$ is true for all positive integers n, the basis step shows that $P(1)$ is true. However, the inductive steps in these two proof methods are different. In a proof by strong induction, the inductive step shows that $P(j)$ is true for all positive integers j not exceeding k, then $P(k+1)$ is true. That is, for the inductive hypothesis we assume $P(j)$ is true for $j = 1, 2, \ldots, k$.

Let's state this principle:

To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete two steps:

Basis Step:

We verify that the proposition $P(1)$ is true.

Inductive Step:

We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

Note that when we use strong induction to prove $P(n)$ is true for all positive integers n, our inductive hypothesis is the assumption that $P(j)$ is true for $j = 1, 2, \ldots, k$. That is, the inductive hypothesis includes all k statements $P(1)$, $P(2)$,..., $P(k)$ to prove $P(k+1)$, rather than just the statement $P(k)$ as in a proof by mathematical induction, strong induction is a more flexible proof technique.

Strong induction is sometimes called the second principle of mathematical induction of complete induction.

Let b be a fixed integer and j a fixed positive integer. The form of a strong induction we need tells us that $P(n)$ is true for all integers n with $n \geq b$ if we can complete these two steps.

1. Basis Step: We verify that the propositions $P(b)$, $P(b+1)$,...,$P(b+j)$ are true.
2. Inductive Step: We show that $[P(b) \wedge P(b+1) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for every integer $k \geq b+j$.

Strong induction can also work in computational geometry.

A polygon is a closed geometric figure consisting of a sequence of line segments s_1, s_2, ..., s_n, called sides. Each pair of consecutive sides, s_i and s_{i+1}, $i = 1, 2, \ldots, n-1$, as well as the last side s_n and the first side s_1, of the polygon meet at a common endpoint, called a vertex. A polygon is called simple if no two nonconsecutive sides intersect. Each simple polygon divides the plane into two regions: its interior, consisting of the points inside the curve, and its exterior, consisting of the points outside the curve.

A polygon is called convex if every line segment connecting, two points in the interior of the polygon lies entirely inside the polygon. A diagonal of a simple polygon is a line segment connecting two nonconsecutive vertices of the polygon, and a diagonal is called an interior diagonal if it lies entirely inside the polygon, except for its endpoints.

One of the most basic operations of computational geometry involves dividing a simple polygon into triangles by adding nonintersecting diagonals. This process is called triangulation.

Theorem 5.1

A simple polygon with n sides, where n is an integer with $n \geq 3$, can be triangulated into $n-2$ triangles.

The validity of both the principle of mathematical induction and strong induction follows from a fundamental axiom of the set of integers, the well-ordering property. The well-ordering property states that every nonempty set of nonnegative integers has a least element.

5.3 Recursive Definitions and Structural Induction

We can prove results about recursively defined sets using structural induction.

We can use two steps to define a function with the set of nonnegative integers as its domain:

Basis Step: Specify the value of the function at zero.

Recursive Step: Give a rule for finding its value at an integer from its values at smaller integers.

Such a definition is called a recursive or inductive definition. note that a function $f(n)$ from the set of nonnegative integers to the set of a real numbers is the same as a sequence a_0, a_1,..., where a_i is a real number for every nonnegative integer i¿

Recursively defined functions are well-defined. That is, for every positive integer, the value of the function at this integer is determined in an unambiguous way.

We can show that the Euclidean algorithm uses $O(\log b)$ divisions to find the greatest common divisor of the positive integers a and b, where $a \geq b$.

Theorem 5.2

Lame's Theorem.

Let a and b be positive integers with $a \geq b$. Then the number of divisions used by the Euclidean algorithm to find $\gcd(a, b)$ is less than or equal to five times the number of decimal digits in b.

Just as in recursive definition of the functions, recursive definitions of sets have two parts, a basis step and a recursive step. In the basis step, an initial collection of elements is specified. In the recursive step, rules for forming new elements in the set from those already known to be in the set are provided. Recursive definitions may also include an exclusion rule, which specifies that a recursively defined set contains nothing other than those elements specified in the basis step or generated by applications of the recursive step.

We can define Σ^*, the set of strings over Σ recursively.

Definition

The set Σ^* of strings over the alphabet Σ is defined recursively by

Basis Step: $\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

Recursive Step: If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Definition

Two strings can be combined via the operation of concatenation. Let Σ be a set of symbols and Σ^* the set of strings formed from symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

Basis Step: If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

Recursive Step: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

Definition

The set of rooted trees, where a rooted tree consists of a set of vertices containing a distinguished vertex called the root, and edges connecting these vertices, can be defined recursively by these steps:

Basis Step: A single vertex r is a rooted tree.

Recursive Step: Suppose that T_1, T_2,...,T_n are disjoint rooted trees with roots r_1, r_2,...,r_n, respectively. Then the graph formed by starting with a root r, which is not in any of the rooted tress T_1, T_2,...,T_n, and adding an edge from r to each of the vertices r_1,r_2,...,r_n is also a rooted tree.

Definition

The set of extended binary trees can be defined recursively by these steps:

Basis Step: The empty set is an extended binary tree.

Recursive Step: If T_1 and T_2 are disjoint extended binary trees, there is an extended binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2 when these trees are nonempty.

Definition

The set of full binary trees can be defined recursively by these steps:

Basis Step: There is a full binary tree consisting only of a single vertex r.

Recursive Step: If T_1 and T_2 are disjoint full binary trees, there is a full binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2.

Instead of using mathematical induction to directly prove results about recurisvely defined sets, we can use a more convenient form of induction known as structural induction.

Basis Step: Show that the results holds for all elements specified in the basis step of the recursive definition to be in the set.

Recursive Step: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the results holds for these new elements.

The validity of structural induction follows from the principle of mathematical induction for nonnegative integers. To see this, let $P(n)$ state that the claim is true for all elements of the set that are generated by n or fewer applications of the rules in the recursive step of a recursive definition. We will have established that the principle of mathematical induction implies the principle of structural induction if we can show that $P(n)$ is true whenever n is a positive integer. In the basis step of a proof by structural induction we show that $P(0)$ is true. That is, we show that the result is true of all elements specified to be in the set in the basis step of the definition. A consequence of the recursive step is that if we assume $P(k)$ is true, it follows that $P(k+1)$ is true. When we have completed a proof using structural induction, we have shown that $P(0)$ is true and that $P(k)$ implies $P(k+1)$. By mathematical induction it follows that $P(n)$ is true for all nonnegative integers n. This also shows that the result is true for all elements generated by the recursive definition, and shows that structural induction is a valid proof technique.

Definition

We define the height $h(T)$ of a full binary tree T recursively.

Basis Step: The height of the full binary tree T consisting of only a root r is $h(T) = 0$.

Recursive Step: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has height $h(T) = 1 + \max(h(T_1), h(T_2))$.

Theorem 5.3

If T is a full binary tree T, then $n(T) \leq 2^{h(T)+1} - 1$.

5.4 Recursive Algorithms

Sometimes we can reduce the solution to a problem with a particular set of input values to the solution of the same problem with smaller input values.

When such a reduction can be done, the solution to the original problem can be found with a sequence of reductions, until the problem has been reduced to some initial case for which the solution is known. For instance, for finding the greatest common divisor, the reduction continues until the smaller of the two numbers of zero, because $\gcd(a, 0) = a$ when $a > 0$.

Definition

An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

A recursive defintion expresses the value of a function at a positive integer in terms of the values of the function at smaller integers. This means that we can devise a recursive algorithm to evaluate a recursively defined function at a positive integer. Instead of successively reducing the computation to the evaluation of the function at smaller integers, we can start with the value of the function at one or more integers, the base cases, and successively apply the recursive definition to find the values of the function at successive larger integers. Such a procedure is called iterative. Often an iterative approach for the evaluation of a recursively defined sequence requires much less computation than a procedure using recursion.

Lemma 5.4

Two sorted lists with m elements and n elements can be merged into a sorted list using no more than $m + n - 1$ comparisons.

Theorem 5.5

The number of comparisons needed to merge sort a list with n elements is $O(n \log n)$.

6 Counting

6.1 The Basics of Counting

6.2 The Pigeonhole Principle

6.3 Permuations and Combinations

6.4 Binomial Coefficients and Identities

6.5 Generalized Permuations and Combinations

7 Discrete Probability

8 Advanced Counting Techniques

8.1 Applications of Recurrence Relations

8.2 Inclusion-Exclusion

9 Relations

10 Graphs

10.1 Graphs and Graph Models

10.2 Graph Terminology and Special Types of Graphs

11 Trees