Theoretical Foundations of Computer Engineering

Chinmay Hegde and Ahmed Kamal

July 8, 2017

Abstract

These are abridged lecture notes for the course, "Theoretical Foundations of Computer Engineering" that we offer at Iowa State University each semester. Topics include basics of modern logic; methods of proofs, including induction; sets, functions, and relations; elementary combinatorics; basics of probability; and an introduction to graph theory. The material in these lecture notes are primarily adapted from the textbook, "Discrete Mathematics and its Applications" by Rosen. Other resources used to construct these notes include "Essentials of Discrete Mathematics" by Hunter, and "Mathematics for Computer Science" by Lehman, Leighton, and Meyer.

Contents

Chapter 1

Overview

The overall goal of this course is to understand how to use mathematical principles in order to solve problems in computer science and engineering.

It is not a stretch to claim that all innovation in mechanical engineering can be distilled down to a (relatively concise) set of laws in *physics*, just as how all innovation in material science can be boiled down to a set of tenets in *chemistry*. Analogously, the behavior of all computing systems can be explained by a few principles in *discrete mathematics*. This course will introduce several such mathematical principles and illustrate their connections to important computer engineering problems. This will be a somewhat different brand of mathematics from what is typically taught in math courses, but just as important.

Chapter 2

Foundations of Logic

A large portion of the course will be devoted to the study of mathematical *proofs*. Proofs can be used to certify that a certain piece of software (or hardware) will **always** work as expected. Such certificates are essential for modern computing systems: with (literally) millions of building blocks in every computer system/computer network/software program, it is vital to ensure that every little block does its job exactly as it should. Proofs enable us to **unambiguously** reason about such systems, detect and correct bugs, and help build faster and better systems.

What is a "proof"? In essence, a "proof" is a method of establishing truth. Now, defining the meaning of 'truth' can be a bit fuzzy, and therefore, the precise meaning of a "proof" differs depending on who you talk to. We will follow the mathematical notion of "proof":

> *A proof of a proposition is a chain of logical deductions leading to the proposition from a set of axioms.*

Quite a mouthful! If we want to give a proof of some proposition, we first need to precisely understand what a "proposition" means, what "axioms" are, and also understand the rules of the game (what types of logical deductions are permitted and what aren't). Today and the next lecture, we will focus on the first bit – propositions.

A bit of backstory: Logical thinking has a rich tradition in *philosophy*, dating back to the Ancient Greeks. Modern logic began to emerge in the 16th century with the work of Descartes and Leibnitz, but became concrete only about 150 years ago when the British mathematicians Boole and De Morgan realized that one can reason about logical statements via *mathematical operations*. This is one of those subtle but remarkable ideas that don't sound like much, but in hindsight, truly changed the course of history. In particular, Lady Ada Lovelace (a student of De Morgan) used these ideas to develop what is now considered the first algorithm that can be executed by a machine, or in other words, the first *computer program*.

The rest of this lecture will introduce some vocabulary in basic logic which we will use for the remainder of the course.

Propositions

A *proposition* is a declarative sentence (or statement) that is either true or false, but not both.

For instance, the following are propositions:

> Ames is a city located in the state of Iowa.
>
> $2 + 2 = 4$.
>
> $2 + 5 < 6$.

The first two statements are true, while the last statement is false. However, all three are legit propositions.

The *truth value* of a proposition is denoted as "T" or "1" if a proposition is true, and "F" or "0" if it is false. For example, the truth values of the above 3 propositions are T, T, and F respectively (or if you like: 1, 1, and 0 respectively.)

Not all statements are propositions. For example:

1. What is your name?
2. Be quiet!
3. It will rain tomorrow.
4. $x + 1 = 6$.
5. Booyah.

The sentence (1) is a question (not a statement), and (2) is an imperative (not a declarative). The next two are indeed declarative statements and are a bit subtle. In the case of (3), the reality is that its truth or falsehood depends on chance (and not determined a priori). In the case of (4), its truth or falsehood depends on what x is. (This kind of statement is called a *predicate*; more on this later in the course.) (5) isn't a sentence `^-^`.

So, depending on whether they are true or not, propositions can be assigned a truth value – just as how a variable can be assigned a value. Therefore, it is instructive to think of statements as *propositional variables*. For example, the variable p can be used to represent the proposition:

> p: It is winter season right now in Ames.

and the *value* of p will be equal to F. Similarly, if we denote q as:

> q: The 2020 Tokyo Olympics are yet to happen.

then the value of q is T.

Unfortunately, it is not always obvious whether a proposition is true or false. For example, the proposition:

> Every map can be colored with four colors so that adjacent regions on the map have different colors.

did not have a conclusive answer one way or the other for several centuries. It was finally proved to be true in 1976 by Appel and Haken, and is now called the "Four Color Theorem".

Logical connectives

Let us get into the habit of thinking of propositions as mathematical variables. The next step is to define a few basic mathematical *operations* that can manipulate their values, combine different propositions, and so on. We call them *connectives*.

Let p and q be propositional variables. We begin with the three most fundamental operations:

- *Negation*: Written as $\neg p$, representing "NOT p".
- *Conjunction*: Written as $p \wedge q$, representing "p AND q"
- *Disjunction*: Written as $p \vee q$, representing "p OR q, OR both."

You may have seen these already in a circuits class; each of these can be realized using a physical device called a "gate".

Let us start with negation, which is the easiest. The operation $\neg p$ simply flips the truth value of p. If p is true (i.e., its value is T), then its negation is false and vice versa. For example, if p represents the proposition "2 + 2 = 5", then the value of p is F; therefore, $\neg p$ has a value of T.

Conjunction is next. Given two propositional variables p and q, the value of $p \wedge q$ is T if both p **and** q have a value of T; in all other cases, the value of $p \wedge q$ is F. For example, if p represents "John likes apples", q represents "John likes oranges", and r represents "John likes apples and oranges", then clearly $r = p \wedge q$, since r can be true only if both p and q are also true.

Disjunction is slightly different. Given two propositional variables p and q, the value of $p \vee q$ is T if **at least one** of p and q have a value of T; if both p and q are F, then the value of $p \wedge q$ is F. For example, if p represents "John has completed ComS 228", q represents "John has enrolled in ComS 228", and r represents "John has the necessary pre-reqs for CprE 310", then clearly $r = p \vee q$,

There is a succinct way to characterize the effect of logical connectives: it is called a *truth* table. Basically, a truth table enumerates all possible combinations of value assignments to the propositional variables, and calculates the output of the connectives. Here is the truth table for the above 3 operations; here, r represents $\neg p$, u represents $p \wedge q$, and v represents $p \vee q$.

p	q	r	u	v
F	F	T	F	F
F	T	T	F	T
T	F	F	F	T
T	T	F	T	T

The use of connectives lets us start with a few base propositions, and build complicated statements in a myriad of ways using these propositions as building blocks. We call them

"compound propositions". For example, expressions like $p \vee (q \wedge r)$ or $\neg(\neg p \wedge \vee q)$ are compound propositions.

One note about using connectives: be mindful of the *order* in which you evaluate them. For example, $p \vee q \wedge r$ is ambiguous, and may give rise to *completely different* values depending on which connective is evaluated first. It is best to demarcate your preference using parentheses: $(p \vee q) \wedge r$ indicates that the disjunction operation between p and q is evaluated first, and the result of that is conjuncted with r. The preferred order is parenthesis -> negation -> conjunction/disjunction.

English to symbolic logic (and vice versa)

Let's now apply some of the principles of logic that we have just learned to represent ordinary sentences in English.

Suppose we want to write down a logical expression for

> *It is sunny but not hot.*

The first thing to do is identify the propositions in this sentence and model them as propositional variables. One choice (not unique!) is as follows:

> p: It is sunny.
>
> q: It is hot.

Therefore, the original sentence can be represented as $p \wedge (\neg q)$.

A sentence like "It is neither sunny nor hot" can be written as $(\neg p) \wedge (\neg q)$. In the opposite direction: if we are given the compound proposition $p \wedge q$, then we can interpret its meaning to represent "It is both sunny and hot".

Let us end with a more interesting example. Suppose we have two propositions:

> p: I will go home via bus.
>
> q: I will go home via car.

How should we represent:

> r: I will go home either via bus or via car.

At first glance, this looks like $p \vee q$, but there is an implicit "not both" in the sentence! Natural spoken language has several such (unspoken) ambiguities, which is not a problem in daily conversation, but can have serious implications if, for instance, you are trying to formulate ideas precisely.

A compound statement like this is called an *exclusive-OR*, and can be written as:

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

or more concisely:

> $p \oplus q$

(Easy homework exercise: can you write down the truth table for $\oplus$?)

Chapter 3

Propositional Logic

Thus far, we encountered:

- propositions, propositional variables, and truth values
- logical connectives (negation, conjunction, disjunction) and compound propositions
- their application for modeling semantics in spoken language.

In this lecture, we will go a bit deeper into these ideas. But first, we serve up some more vocabulary.

Special compound propositions

A compound proposition is called a *tautology* if its truth value is T under any assignment of truth values to its components.

For example,

$p \vee (\neg p)$

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

are both tautologies.

An English example: if p represents the proposition "It is sunny outside", then $p \vee (\neg p)$ is the same as saying "It is sunny outside, or it isn't sunny outside", which is obviously a tautology.

The opposite of a *tautology* is a *contradiction*; its truth value is F under any assignment of values to its components. For example,

$p \wedge (\neg p)$

is a contradiction. Again, an English example: if p represents the proposition "This apple is red", then $p \wedge (\neg p)$ is the same as saying "This apple is red and is not red", which is a contradiction.

Many propositions are neither tautologies nor contradictions: in such cases, their truth value depends on the states of the internal variables, and are called *contingencies*.

The surest way to check whether something is a tautology (or a contradiction) is to evaluate its *truth table*, and check whether all the output values are T (or F).

Conditionals

You have probably seen the logical construct "if `<assume something>` then `<something else happens>`" in math proofs. You have also probably used an "if-then" command when writing computer programs.

Modern logic codifies this into a special connective. A proposition of the form "if p then q", represented as "$p \implies q$" (and read aloud as "p implies q") is called a *conditional proposition* or an *implication*. Here, p is called the hypothesis (or premise, or assumption), and q is called the conclusion (or consequence.)

For example:

> *If n is divisible by 4, then n is divisible by 2.*
>
> *If John lives in Ames, then John is a resident of Iowa.*

are both conditionals.

This seems quite intuitive; however, **there is a catch** (and this trips up people all the time.) The convention is that $p \implies q$ is always assigned T, **except** when p is true and q is false. Let us write down the truth table for " $p \implies q$" :

p	q	p $\implies$ q
T	T	T
T	F	F
F	T	T
F	F	T

The above truth table indicates that the following propositions are *also* true:

1. *If time travel is possible, then 2 < 4.* (F $\implies$ T)
2. *If the sun rises in the west, then I have discovered a cure for cancer.* (F $\implies$ F).

These are somewhat bizarre statements and you will never encounter them in any reasonable setting, but they are worth taking a closer look.

In (1) above, since the conclusion "2 < 4" is clearly true, *it doesn't matter* whether the hypothesis is true or not; we are in either the 1st or 3rd line of the truth table, and the overall proposition is true!

The case (2) above is even more strange; since the hypothesis is clearly false, *it doesn't matter* whether the consequence is true or not; we are in the 3rd or 4th line of the truth table, and the overall proposition is true!

On the other hand, this is false:

If $2 + 2 = 4$*, then New York is in Canada.*

The premise is true, but the consequence is not; therefore, the overall statement is false!

You may wonder why implications having false hypotheses can themselves be true. A different way to think about it is as follows. Consider the following example (adapted from Enderton's book on logic):

If you are telling the truth, then pigs can fly.

Here p represents "You are telling the truth" and q represents "pigs can fly". From the truth table for $\implies$, we assign a value of T to this proposition whenever you are fibbing. This doesn't meet that you stretching the truth will suddenly *cause* pigs to sprout wings and start flying. Instead, it makes an assertion about pigs **provided** a certain hypothesis is met. If this hypothesis is false, then then the statement is said to be *vacuously true.*

Other "Common English" phrases that are the same as "$p \implies q$" are:

1. *if* p *then* q
2. *In order for* q *to be true, it is sufficient for* p *to occur.*
3. p *is a sufficient condition for* q*.*
4. p *is true only if* q *is true.*

Biconditionals

The proposition $p \iff q$ (read as "p if and only if q") is a *biconditional* implication; it is true only when both p and q have the same truth values, i.e., when they are both simultaneously true or simultaneously false.

In ordinary math, this definition encodes the (commonly used phrase) "p is a necessary and sufficient condition for q".

Logical equivalence

Compound propositions can become rather complicated, and it is often prudent to replace one proposition with a simpler one. Of course, you can only do that if they are functionally the same. For example, we already saw that p and $\neg(\neg p)$ have the same truth assignments, regardless of what p is.

As a second example, you can check that $p \implies q$ and $\neg p \vee q$ have the same truth table:

p	q	$p \implies q$	$\neg p \vee q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

Compound propositions whose truth values are the same, regardless of their internal variables, are called *logically equivalent*. The symbol for logical equivalence is $\equiv$.

Exercises: check (using truth tables) that the following are logically equivalent:

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

These are called *De Morgan's* Laws for Logic.

- $p \iff q \equiv (p \implies q) \wedge (q \implies p)$

In words, "p is a necessary and sufficient condition for q" is the same as saying "p implies q and q implies p". This is a technique that we will often use in proofs later on in the course.

Converse, contrapositive, inverse

Do these propositions say the same thing?

1. *If I am very thirsty, then I feel dizzy.*
2. *If I do not feel dizzy, then I am not very thirsty.*

Let p denote "I am thirsty" and q denote "I am tired". Then, the two propositions can be written as "$p \implies q$" and $(\neg q) \implies (\neg p)$. Writing out the truth table, we get:

p	q	$p \implies q$	$(\neg q) \implies (\neg p)$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

In other words, (1) and (2) are *logically equivalent*. A statement of the form $(\neg q) \implies (\neg p)$ is called the *contrapositive* of the implication $p \implies q$. They are just two different ways of saying the same thing. Again, this is extremely useful in proofs; instead of trying to proving a given implication, we can instead prove its contrapositive, and rest assured that we are done.

On the other hand, how about:

3. *If I feel dizzy, then I am very thirsty.*

This is the proposition $q \implies p$, and is called the *converse* of $p \implies q$; even at face value, this seems like a rather different proposition than $p \implies q$ (and can be checked again using the truth table.)

Finally, consider:

4. *If I do not feel thirsty, then I do not feel dizzy.*

the proposition $\neg p \implies \neg q$ is called the *inverse* of $p \implies q$. Again, this is a *different* proposition than $p \implies q$, checked via enumerating the truth table.

Some applications

Again, let us apply some of the principles we learned.

How should we write out the natural language statement:

> You can take CprE 409 only if you have taken either CprE 301 or EE 324.

Denote r as "You can take CprE 409", p as "You have taken CprE 301", and q as "You have taken EE 324". Then, the above expression is:

$$r \implies p \vee q.$$

Here is a second, real(istic) application that arises in model-based system design. Design specifications are sometimes given as a series of *rules*, say:

> If system sensors encounter phenomenon C_i then perform action A_i for i=1,. . . ,n.

Whether or not a system is acting according to the *overall* specification can be written as the logical expression:

$$(C_1 \implies A_1) \wedge (C_2 \implies A_2) \wedge \ldots \wedge (C_n \implies A_n)$$

Now say only conditions C_1 and C_7 occur. Then if the system does indeed take actions A_1 and A_7, the quantities $(C_1 \implies A_1)$ and $(C_7 \implies A_7)$ are both equal to T; the rest of the implications are also T *vacuously*. Therefore, the overall system is behaving according to specification.

On the other hand, if C_7 occurs and the system does *not* take action A_7, then the quantity $(C_7 \implies A_7)$ is false and the system is not acting according to the overall specification.

Chapter 4

Predicates

By now, we have had an initial taste of what propositional logic is all about. We have seen how to model declarative sentences as propositions, and discussed ways to manipulate propositional variables using a number of logical connectives. We also discussed the concept of *logical equivalence* of different propositional expressions.

Today, we will discuss a slightly more general system of logic known as *predicate logic*. But more on that in a moment; first, a few more concepts and applications of propositional logic.

Formal checking/verification

Recall that some propositions are tautologies: a tautology is true no matter what truth values its variables may have. (Similarly, some propositions are contradictions: they are always false.) For example,

$$p \vee \neg p$$

is a tautology no matter what the value of p is. Tautologies can be thought of as logical expressions that capture certain fundamental truths.

How do we verify whether something is a tautology? The surest (and brute-force) way to do this is to write out a truth table. For example, if we want to show that

$$s := (p \wedge q) \implies (p \vee q)$$

is a tautology, then we simply write out the truth table:

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

and verify that the last column is all "T".

Of course, this procedure is somewhat cumbersome. The above example only used 2 propositional variables (p and q). For a proposition involving n variables, the truth table will have 2^n lines (**Exercise: Why?**) and for even small logical systems (with, say, $n = 30$ variables), we are already talking about *billions* of lines to evaluate.

An alternate approach that *sometimes* helps is to appeal to logical equivalence. We already saw some examples of logical equivalences.

- $(p \implies q) \equiv (\neg p \vee q)$
- $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
- $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

Recall from previous lecture that the last two are called "De Morgan's Laws". Of course, there are many more that we can use: all axioms/properties/theorems of Boolean logic that you learned in CprE 281 are perfectly OK to be used as logical equivalences.

We can use some combination of these equivalences to simplify the expression. For example:

$$\begin{aligned} s &:= (p \wedge q) \implies (p \vee q) \\ &\equiv \neg(p \wedge q) \vee (p \vee q) \\ &\equiv (\neg p \vee \neg q) \vee (p \vee q) \\ &\equiv (\vee p \vee p) \vee (\neg q \vee q) \\ &\equiv T \vee T \\ &\equiv T. \end{aligned}$$

The above sequence of arguments can be viewed as a *proof* that the original expression s is a tautology. It is not always obvious which particular sequence of logical equivalences should be used to prove that something is a tautology, but that comes with a bit of practice.

Satisfiability

Many propositions are neither tautologies nor contradictions: their truth value depends on the assignment of truth values to their internal variables. Such statements are known as *contingencies*. If there is *at least one* assignment of truth values to its variables that makes the overall expression true, then the expression is called *satisfiable*.

If p is not satisfiable, then p is (by definition) a contradiction and $\neg p$ is a tautology. A different way to define satisfiability is as follows: p is unsatisfiable if and only if $\neg p$ is a tautology.

As before, we can ask the same question: how to check whether a compound proposition is satisfiable? Again, the brute-force (but safest) method is via truth tables. For example, if we want to check whether:

$$s := (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

is satisfiable, we can write out the truth table (consisting of $2^3 = 8$ rows) and verify that s is satisfiable if p,q,r are either all true or all false, i.e., $p \equiv q \equiv r$.

The alternative is to simplify the expression via inspection. Since s is a conjunction of 3 propositions, for s to be true *each* of the sub-expressions $(p \vee \neg q), (q \vee \neg r), (r \vee \neg p)$ should be true. Therefore, if p is true then both r and q have to be true. and if p is false, then both q are r are false.

Exercise: verify that:

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

is *unsatisfiable*.

In fact, the general problem of checking whether an expression is satisfiable is called the *SAT* problem. Since the truth table contains 2^n rows, the brute-force algorithm to solve the *SAT* problem is said to have *exponential* running time. A famous open problem in computer science is whether there is any algorithm that runs substantially faster (i.e., with polynomial running time.)

Predicates

Not all statements are propositions *per se*. For instance:

x + 1 = 6

is true if $x = 5$, and false otherwise. Such statements are called *predicates*. We have already (briefly) discussed predicates before; if propositions can be modeled as variables, then predicates are analogous to *functions*.

A predicate, or a *propositional function*, is a statement whose truth value depends on one or more variables. We use the notation:

P(x): x + 1 = 6

and say that the *truth value* of *P(x)* depends on x. As another example, if we define:

Q(n): n is a perfect square

then *Q(1), Q(4), Q(25), Q(100)* are all true, while *Q(23)* is false.

Similarly, let:

P(x,y): x = y + 2.

Then, *P(4,2)* is true and *P(1,2)* is false.

The variable in a given predicate is assumed to possess values from a certain (unspecified) set called the *domain of discourse*. Consider:

P(n): n is a prime.

P(σ): σ contains at most 1 vowel.

In the first example, the domain of discourse is the set of positive integers. In the second, the domain of discourse can be the set of all words (or strings). Whenever you come across a predicate, it is important and useful to clearly identify the universe of discourse.

Quantifiers

Predicates can be true or false; some predicates are always true and some always false. a *quantifier* expresses the *extent* to which a predicate is true.

There are two main types of quantifiers. Given a predicate $P(x)$, consider the assertion:

"For all x, *P(x)* is true"

is an assertion that can be concisely written as:

$\forall x \ P(x)$

The symbol $\forall$ is called a *universal quantifier*. Observe that application of the quantifier has transformed the predicate *P(x)* to a *proposition* which has a specific truth value. For example,

$\forall x, \ x^2 \geq 0$

is a *proposition* that is true if the domain of discourse is the set of real numbers. Similarly, the

$\forall x, \ x + 1 = 6$

is false (it is only true if x is equal to 5). Synonyms for "for all x" include "for any x", "for each x", and "for every x".

On the other hand, consider the assertion:

"For some x, *P(x)* is true"

concisely written as:

$\exists x \ P(x)$

The symbol $\exists$ is called an *existential quantifier*. It stipulates that there is at least one element in the domain of discourse for which the predicate is true.

For example,

$\exists A$, A knows how to program in Java

is true if the domain of discourse is the set of CprE 310 students. On the other hand,

$\exists x, \ x + 1 = 6.5$

is true if the domain of discourse is the real numbers (the choice $x = 5.5$ makes it true), but false if the domain of discourse is the natural numbers.

Exercises: Convince yourself that the propositions

- $\forall x, (x + 1)^2 = x^2 + 2x + 1$
- $\forall x, \quad x^2 \geq x$
- $\exists x, \quad x^2 = \pi$
- $\exists x, \quad x + 1 < x$

are true, false, true, and false respectively, assuming that the domain of discourse is the set of real numbers.

We can also *combine* universal and existential quantifiers. As an example: if we wish to express the proposition:

For every prime number, there exists a prime number that is larger.

using predicate logic, we can write it as:

$\forall p \; \exists q \;\; p < q$

where the domain of discourse is the set of prime numbers. (In fact, this is a *true* proposition.)

Just as how the order of logical connectives is important, the order of writing down quantifiers is also important! In the above example, if we wrote down:

$\exists q, \; \forall p, \;\; p < q$

then this would represent the statement:

There exists a prime number that is larger than every prime number.

which is a completely different statement! (one that is obviously false.)

Exercise: Suppose *P(x,y)* represents the predicate:

P(A,B):= A and B are related

where the domain of discourse is the set of all people in the world. Which of the following statements is true and which one is false?

$\forall A, \; \exists B, \;\; P(A, B)$

$\exists A, \; \forall B, \;\; P(A, B)$

Chapter 5

Rules of inference

In Lecture 1, we mentioned that a major objective of 310 is to develop a systematic framework that lets us prove mathematical statements. Recall the mathematical definition of a "proof":

> *A proof of a proposition is a chain of logical deductions leading to the proposition from a set of axioms.*

The operating terms in this definition are "proposition", "axioms", and "chain of logical deductions". We understand what propositions are. We have seen certain types of logical "deductions" but will dig a bit deeper into this; specifically, we will define what types of deductions are kosher within our framework. We also need to properly define what an "axiom" is. But first, we need some more vocabulary.

Arguments

A *axiom* is a proposition that is assumed to be (unquestionably, universally) true.

A *theorem* is a proposition that can be proved to be true.

An *argument* is a sequence of propositions that starts from an proposition (or set of propositions) and ends in a final proposition. In any argument, all propositions (except the last one) are called *premises*; the final proposition is called the *conclusion*. A *hypothesis* is a proposition that is assumed to be true in the context of a particular argument.

A proof is a *valid* (i.e., logically sound) argument that begins with propositions that are *true* (i.e., hypotheses or axioms) and leads to a *true* conclusion.

As a simple example, consider the following "proof" that John is eligible for CprE 310.

> If John has taken ComS 228, then John is eligible to enroll in CprE 310. (**premise**)
>
> John has taken ComS 228. (**hypothesis**)
>
> Therefore, John is eligible to enroll in CprE 310. (**conclusion**)

This flow of logic seems fairly obvious and intuitive. However, here is a way from first principles (without relying on intuition) to check whether this is logically sound.

Let p denote "John has taken ComS 228" and q denote "John is eligible for CprE 310". Then, the above reasoning can be represented symbolically using the following notation:

$$\begin{array}{l} p \implies q \\ \underline{\quad p \quad} \\ \therefore\ q \end{array}$$

This is an example of an *argument*. In order to elevate this into a *proof* (or valid argument) of John's eligibility, we need to assign a value of "T" to *each* of the premises (everything above the horizontal line) and check that this leads to a value of "T" to the conclusion (the statement below the horizontal line). However, this is easy to see: assigning a value of "T" to both p as well as $p \implies q$ forces q to be also "T", recalling the property of " $\implies$ ".

Note: We could have expressed the above argument purely using propositional logic:

$$((p \implies q) \wedge p) \implies q$$

One can check that this logical statement is a tautology; in particular, if when both $p \implies q$ as well as p are true, then q has to be true. But the 3-line notation as used above makes the flow of logic very clear.

Here is a more abstract (and tedious) example, albeit one that illustrates the steps involved in constructing an argument. Consider 3 premises:

(1) $p \implies q$

(2) $r \implies s$

(3) $q \wedge s \implies z$

Assuming that the above 3 premises are true, and also assuming that (4) p is true, and (5) r is true, arrive at the conclusion that z is true.

Here is the argument. In the previous example, we showed that p being true, along with Premise (1) being true, leads to q also being true. In symbols,

$$\begin{array}{l} p \implies q \\ \underline{\quad p \quad} \\ \therefore\ q \end{array}$$

By an identical reasoning, since r is true and Premise (2) is true, we also have that s is true:

$$\begin{array}{l} r \implies s \\ \underline{\quad r \quad} \\ \therefore\ s \end{array}$$

Combining q being true and s being true, we get $q \wedge s$ being true:

$$\begin{array}{l} q \\ s \\ \hline \therefore\ q \wedge s \end{array}$$

Finally, combining $q \wedge s$ being true and Premise (3), we get the desired conclusion.

$$\begin{array}{l} q \wedge s \implies z \\ q \wedge s \\ \hline \therefore\ z \end{array}$$

This may seem like an arcane sequence of operations that magically led us to our desired conclusion, but the fact of the matter is that we simply used all the available tools at our disposal. One way to look at this is to work backwards from the desired result: showing that z is true would require showing that q and s are individually true (from Premise (3)), and this can be shown using Premises (1) and (2).

Truth tables

How do we check whether or not an argument is valid? As we discussed before in the case of propositions and predicates, the surest way is via truth tables. We construct a truth table of all possible assignments to internal variables, and try to look for *critical rows* where every hypothesis, premise, and conclusion is assigned a value of "T". If such a row exists, then the argument is valid; if no such row exists, then the argument is invalid.

Here is an example. Consider the argument:

> If I am sleepy, then I want to drink coffee.
>
> It is true that I am sleepy.
>
> Therefore, I feel sleepy and I want to drink coffee.

Symbolically, we can write this as:

$$\begin{array}{l} p \implies q \\ p \\ \hline \therefore\ p \wedge q \end{array}$$

Again, this argument seems intuitively obvious. But how can we check this intuition? Let us build the truth table for all the 3 lines of the argument:

p	q	$p \implies q$	$p \wedge q$
T	T	**T**	**T**
T	F	F	F
F	T	T	F
F	F	T	F

The first row shows the existence of a truth assignment where each of the 3 lines is assigned a value of "T". Therefore, the argument is valid!

Rules of inference

In order to avoid the brute-force approach of truth tables, we can alternatively employ some standard logical deductions, or *rules of inference*. These are all straightforward adaptations of propositional logic that we have already seen before, but written in a somewhat different (and strange-looking) notation.

There are several rules of inference that we can use in logical arguments. Here, we list a few commonly used ones. The validity of the rules can be checked via truth tables and we leave this as an easy exercise. In each of the rules below, think of p, q, r etc. as being propositional variables.

Modus ponens. We have seen this one earlier in this lecture:

$$\begin{array}{l} p \implies q \\ \underline{p \qquad\qquad} \\ \therefore\ q \end{array}$$

An example is:

> If it is snowing, I drive to work.
>
> It is snowing.
>
> Therefore, I drive to work.

By the way, the phrase "modus ponens" stands for "method of affirmation", and is a great example of:

> *"omnia dicta fortiora si dicta Latina"*

which roughly translates to

> *Everything sounds cooler when said in Latin.*

Modus tollens. Also seen previously in propositional logic, closely related to the contrapositive:

$$\begin{array}{l} p \implies q \\ \underline{\neg q \qquad\qquad} \\ \therefore\ \neg p \end{array}$$

Example: "If the sun is out, then it is daytime. It is not daytime. Therefore, the sun is not out."

Hypothetical syllogism

$$\begin{array}{l} p \implies q \\ \underline{q \implies r \qquad\qquad} \\ \therefore\ p \implies r \end{array}$$

Another easy one to understand. In words: "If it is raining, then we play indoors. If we play indoors, then we are shooting hoops. Therefore, if it is raining, then we are shooting hoops."

Disjunctive syllogism

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

In words: "Either 1027 is composite or prime. 1027 is not prime. Therefore, 1027 is composite."

Rule of addition. In general symbolic form, we write this as:

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

An example of this rule is the following sentence:

> "It is snowing. Therefore, it is either raining or snowing".

Rule of simplification. We write this as:

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

An example is the following:

> "31 is prime and odd. Therefore, 31 is prime."

Rule of conjunction. Exactly as it sounds:

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

In words:

> "This cup of coffee is free. This cup of coffee tastes good. Therefore, this cup of coffee is free and tastes good."

Rule of resolution

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

In words: "It is snowing or Bart is playing soccer. It is not snowing or Colin is at a ski resort. Therefore, either Bart is playing soccer or Colin is at a ski resort".

This is an incomplete set of inference rules, and you can use others. Indeed while constructing a complicated proof in a future class, you may wonder whether some intermediate statement may be assumed as an fact (or hypothesis), or should be obtained as a conclusion starting from a simpler hypothesis or axiom. There is no clear answer to this, but it is good practice to **clearly** state your assumptions up front.

But these eight rules will be enough for 310. Memorize them!

We end this lecture with four **exercises**. First, is this a valid argument?

> "If today is Thursday, then John will go to the gym." "Today is Thursday." "Therefore, John will go to the gym."

Yes! The argument is valid, and follows the *modus ponens* rule.

Next example. How about:

> "If I play Pokemon Go all day, I won't finish Homework 1. If I don't finish Homework 1, I won't ace the exam. Therefore, if I play Pokemon Go all day, I won't ace the exam."

Valid argument! This is an example of a hypothetical syllogism.

Third example. What about:

> "If I do every practice problem set in 310, I will become an expert in logic. I am an expert in logic. Therefore, I did every practice problem set in 310."

This is a *fallacy*. It resembles modus ponens, but really, the argument is logically represented as:

$$\begin{array}{l} p \implies q \\ \underline{q \qquad\qquad} \\ \therefore\ p \end{array}$$

which is fallacious, since the premises being true need not imply that the conclusion is true. (**Exercise**: Can you verify this claim using a truth table?) This type of incorrect reasoning is unfortunately all too common, and is called the *fallacy of affirming the conclusion.*

Last one. What about:

> "If I do every problem in the book, I will become an expert in logic. I didn't do every problem in the book. Hence, I am not an expert in logic."

Another *fallacy*. This one looks like modus tollens, but in reality, the representation of this argument is:

$$\begin{array}{l} p \implies q \\ \underline{\neg p \qquad\qquad} \\ \therefore\ \neg q \end{array}$$

which is also fallacious (Again, as an **exercise** verify this claim using a truth table.). Such an incorrect argument is called the *fallacy of denying the hypothesis.*

Chapter 6

More rules of inference, introduction to proofs

Recall that instead of using the brute-force truth table approach for showing the validity of an argument, we can instead use *rules of inference*. We have discussed eight rules of inference that involve different combinations of propositions and connectives. Let us now see how to synthesize these rules to construct more complex arguments.

Some applications of rules of inference

Consider the following argument:

> I eat pizza or I eat pasta for dinner. If I eat pizza for dinner, then I fall asleep early. I don't fall asleep early. Therefore, I eat pasta.

Denoting p as "I eat pizza for dinner", q as "I eat pasta for dinner", and "r" as "I fall asleep early". Then, the premises (information that is given) in the above argument are:

1. $p \vee q$
2. $p \implies r$
3. $\neg r$

and the conclusion is:

- q

Assuming that the premises are true, how do we arrive at the conclusion? Here is a 2-step proof. First, combining (1) and (3) above using *modus tollens*, we have:

$$\begin{array}{l} \neg r \\ \underline{p \implies r} \\ \therefore \neg p \end{array}$$

Therefore, $\neg p$ is true. Combining this with (2) using *disjunctive syllogism*, we have:

$$\begin{array}{l} \neg p \\ \underline{p \vee q} \\ \therefore q \end{array}$$

In other words, the conclusion q, i.e., "I eat pasta" is true.

Rules of inference also enable us to solve logic *puzzles* in a systematic manner. Here is an example. Suppose you need to determine the location of a pot of gold that is hidden somewhere in a house. A genie gives you the following information about the house:

1. The house is next to a lake.
2. If the house is next to a lake, then the pot isn't in the kitchen.
3. If the tree in the front yard is an oak, then the pot is in the kitchen.
4. The tree in the front yard is an oak or the pot is buried under the porch.

Where is the treasure?

If you guessed that the treasure is buried under the porch, then you are now a very rich person `^-^`. However, let us be sure that we are correct, and prove this using first principles:

Define propositional variables:

p: The house is next to a lake.

q: The pot is in the kitchen.

r: The tree in the front yard is an oak.

s: The pot is buried under the porch.

From the information that the genie gives us, the following propositions are true:

- p
- $p \implies \neg q$
- $r \implies q$
- $r \vee s$

Now, we need to carefully construct an argument about the location of the pot using rules of inference. Here is the sequence of deductions:

Modus ponens:

$$\begin{array}{l} p \\ \underline{p \implies \neg q} \\ \therefore \neg q \end{array}$$

Modus tollens:

$$\begin{array}{l} \neg q \\ \underline{r \implies q} \\ \therefore \neg r \end{array}$$

Disjunctive syllogism:

$$\begin{array}{l} \neg r \\ r \vee s \\ \hline \therefore\ s \end{array}$$

By the above argument, s is true, i.e., the pot is buried under the porch.

Again, the reason *why* we used this particular sequence of rules isn't obvious, and becoming good at this kind of reasoning comes with practice!

Rules of inference for quantified statements

Similar to rules of inference for propositions, we can also develop rules of inference involving predicates and quantifiers. Recall that we defined two quantifiers for predicates: the *universal* quantifier ($\forall$) and the *existential* quantifier ($\exists$). Fortunately, there are only four such rules that we focus on in 310.

Universal instantiation

If $\forall x, P(x)$ is true, then $P(a)$ is true for each element a in the domain of discourse. In symbols:

$$\frac{\forall x,\ P(x)}{\therefore P(a) \text{ for arbitrary } a}$$

For example, if we have the predicate $P(x) : x^2 \geq 0$, $\forall x, P(x)$ is true if the domain of discourse is the real numbers. Therefore, by instantiating this with $a = -2$, we can derive that $P(-2)$ is true.

Existential instantiation If $\exists x, P(x)$ is true, then $P(a)$ is true for some specific element a in the domain of discourse.

$$\frac{\exists x,\ P(x)}{\therefore\ P(a) \text{ for specific } a}$$

For instance, if we have $P(x) : x^2 = 2$, then $\exists x,\ P(x)$ implies that we can infer the existence of at least one element a for which $P(a)$ is true. (This follows basically from the definition of $\exists$).

Universal generalization This is nothing but universal instantiation in the reverse direction. If we take an arbitrary (generic) element a from the domain of discourse and can show that $P(c)$ is true, then $\forall x, P(x)$ is true. This is written as:

$$\frac{P(a) \text{ for arbitrary } a}{\therefore\ \forall x,\ P(x)}$$

For instance, suppose the domain of discourse is the real numbers, and suppose we start with a generic real number x and use algebra to prove that $(x+1)^2 = x^2 + 2x + 1$. Then, we can use universal generalization to claim that $\forall x, P(x)$ is true.

Existential generalization Existential instantiation in the reverse direction. If we know one element a in the domain for which $P(a)$ is true, then $\exists x, P(x)$ is also true.

$$\frac{P(a) \text{ for specific } a}{\therefore\ \exists x,\ P(x)}$$

We now have a suite of rules of interference for quantified predicates. Keep in mind that we can always do a mix-and-match between propositional rules of inference and predicate rules of inference to synthesize a valid argument. Let us see a simple example. Suppose we start with two premises:

All students who are taking 310 have taken 281.

John is a student who is taking 310.

Use predicate logic to conclude that:

John has taken 281.

As always, we start with defining the predicates:

D(x): x is a student in 310.

C(x): x has taken 281.

The premises can be written in terms of these predicates as follows:

1. $\forall x(D(x) \implies C(x))$
2. $D(John)$ is true.

Now, by universal instantiation, we can derive:

3. $D(John) \implies C(John)$

Finally, combining (3) and (2) using modus ponens, we can derive:

$\therefore\ C(John)$ is true

which is precisely what we set out to show.

Introduction to proofs

We have discussed (in considerable detail) how arguments are constructed. Recall that a proof is a *valid* argument, i.e., a sequence of correctly applied rules of inference that start with a bunch of premises and arrive at a conclusion.

Of course, the key in any proof is to figure out the *precise* sequence of steps needed to arrive at the conclusion. There is no magic recipe for this, unfortunately (other than try out all possible truth assignments and rules of inference, which is more or less what a truth table does.) However, there are certain common *strategies* that mathematical proofs often use. We

will see a few of them today and in future lectures. The goal is to put together (and become familiar with) a toolbox of proof techniques that can be used to prove a variety of theorems; in fact, much of the rest of the course will involve doing precisely this. Let us start with the simplest and most standard techniques.

Direct proofs

A *direct* proof of a conditional statement $p \implies q$ follows by assuming that p is true, and constructing a series of deductions to show that q must also be true.

For example, suppose we have the following "theorem":

/If $x > 2$ and $y > 7$, then $2x + y > 11$./

A direct proof of this theorem would look like this. Suppose we assume that for some generic x and y:

1. $x > 2$
2. $y > 7$

are both true. Then, we can infer from the first premise that

3. $2x > 4$

is true. Adding the inequalities (2) and (3) term by term, we get that

4. $2x + y > 11$

is true. Therefore, the given theorem is true.

Similarly, if we want to prove the statement:

If a given integer n is odd, then n^2 is odd.

A direct proof would be as follows. Assume that the premise is true, i.e., n is an odd integer. Recall that n is odd if and only if it can be written as $n = 2k + 1$ for some integer k. Therefore,

$$\begin{aligned} n^2 &= (2k+1)^2 \\ &= 4k^2 + 4k + 1 \\ &= 2(2k^2 + 2k) + 1 \end{aligned}$$

Therefore, n^2 can also be written as one greater than twice some integer. Thus, n^2 is also odd.

Indirect proofs

An *indirect* proof, or a proof by *contraposition*, of a conditional statement $p \implies q$ consists of proving the contrapositive of the desired conditional, i.e, instead of proving $p \implies q$, we prove $\neg q \implies \neg p$. (Recall that an implication is logically equivalent to its contrapositive.)

For example, suppose we have the following theorem:

If $x + y > 9$, then $x > 2$ or $y > 7$.

In symbols, we need to prove that:

$$x + y > 9 \implies (x > 2) \vee (y > 7).$$

which is of the form $p \implies q$. Suppose q is false, i,e,

$$\neg((x > 2) \vee (y > 7)).$$

is true. By De Morgan's Law, we can simplify this expression as:

$$(x \leq 2) \wedge (y \leq 7)$$

is also true.

By applying the rule of *Simplification* to this expression, we obtain that

$$x \leq 2$$

$$y \leq 7$$

are both true. Adding these inequalities term by term, we get:

$$x + y \leq 9$$

which is nothing but $\neg p$.

Chapter 7

Methods of proof

A good proof is like a well-written software program. The pieces that you need to put together may not be evident right from the beginning, and takes practice and experience. We will try to build an arsenal of proof methods that appear time and time again.

Here is a list of some simple proof methods. We already discussed the first two types in the previous lecture.

Direct proofs

Direct proofs of conditional statements (i.e., propositions of the form $p \implies q$) are constructed as follows: assume that p is true, and show using a sequence of rules of inference that q is true.

For example, suppose we want to prove the following statement:

> Let a, b be positive integers. If $n = ab$ then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.

A direct proof would proceed as follows. The above statement is an implication of the form $p \implies q$, where:

- p: $n = ab$
- q: $(a \leq \sqrt{n}) \vee (b \leq \sqrt{n})$

Assume that the antecedent p is true, i.e., $n = ab$. There are exactly two cases: we have either $a \geq b$ or $a < b$.

If $a < b$, then $a^2 < ab$ by multiplying both sides by a. Therefore, $a^2 < n$, or $a < \sqrt{n}$. Therefore, the consequent q is true.

If $a \geq b$, then $a^2 \geq ab$ by multiplying both sides by a. Therefore, $a^2 \geq n$, or $a \geq \sqrt{n}$. This implies that $1/a \leq 1/\sqrt{n}$; equivalently, $b = n/a \leq \sqrt{n}$. Therefore, the consequent q is true.

In either case, q is true, and therefore the implication is true.

Proofs by contraposition

In contrast, a proof by contraposition of a conditional statement is an *indirect* method of proof: assume that q is false (i.e., $\neg q$ is true), and conclude that p is false, (i.e.,)$\neg p$ is true). We have seen before that $p \implies q$ is logically equivalent to $\neg q \implies \neg p$. Therefore, this is equivalent to proving that $p \implies q$ is true.

Again, let us try and prove the statement:

> Let a, b be positive integers. If $n = ab$ then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.

An indirect proof would proceed as follows. Let p and q be the same as above. Assume that the consequent q is false, i.e., $\neg q$ is true. By De Morgan's Laws, we have:

$$(a > \sqrt{n}) \wedge (b > \sqrt{n})$$

Therefore, $ab > \sqrt{n} \cdot \sqrt{n}$, or $ab > n$. In particular, $ab \neq n$, and hence p is false. This proves that the implication $\neg q \implies \neg p$ is true, or equivalently, $p \implies q$ is true.

Vacuous proofs

A vacuous proof of a conditional statement is constructed by observing that $p \implies q$ is true whenever p is false.

For example:

> *Let* x *be a real number. If* $x^2 + 1 < 0$*, then* $x = 3.14159$*.*

A "proof" of this implication would proceed as follows. Observe that the antecedent (p) in the implication is the proposition $x^2 + 1 < 0$, which is false for all real numbers. Therefore, the above implication is *vacuously true*. Done!

Vacuous proofs are not frequently encountered in pure math, but often arise in logical thinking. Also, when we will study *mathematical induction* later in the course, we will see that vacuous proofs are frequently used to establish proofs of base cases.

Trivial proofs

A trivial proof of a conditional statement is constructed by observing that $p \implies q$ is true whenever q is true, regardless of the truth value of p.

For example:

> *Let* x *be a real number. If* $x > 0$*, then* $x^2 + 1 > 0$*.*

A proof of this implication would be as follows. For any real number x,

$$x^2 + 1 > x^2 \geq 0.$$

Therefore, $x^2 + 1 \geq 0$ and the implication is *trivially* true.

Note that we never used the hypothesis that $x > 0$ in our proof. Still, the stated theorem is true.

Chapter 8

Methods of proof (continued)

We have discussed a few different proof techniques for proving implications of the form $p \implies q$, namely:

- Direct proofs (assume p is true, show that q is true)
- Proof by contraposition (assume)
- Trivial proofs
- Vacuous proofs

We round off our discussion with a list of several additional proof techniques.

Proof by equivalence

Suppose we wish to prove a *biconditional* statement, i.e., any statement of the form "p if and only if q". Recall that $p \iff q$ is logically equivalent to

$(p \implies q) \wedge (q \implies p)$.

Therefore, an if-and-only-if (or an *iff* for short) can be established by proving the forward implications ($p \implies q$) and the reverse implication ($q \implies p$) separately, using any of the techniques for proving implications that we discussed earlier.

This reasoning can be extended to any chain of if-and-only-if statements. Suppose we need to proof that "A iff B iff C". Then, we can arrive at this proof by individually showing that $A \implies B$, $B \implies C$, and $C \implies A$.

Proof by contradiction

A proof by contradiction is a type of indirect proof that is somewhat more general than a proof by contraposition. Proofs by contradiction are structured as follows. To prove that a statement A is true, you assume, to the contrary, that A is false (i.e., $\neg A$ is true). Then, as in a direct proof, you apply a series of logical deductions until you arrive at a statement that

you *know* cannot be true (i.e., a contradiction.) Therefore, the original assumption that A is false is incorrect, and A itself has true.

Proofs by contradictions are an example of the Latin phrase:

Reductio ad absurdum

or "reduction to absurdity". One shows that a statement is true by showing that its negation leads to a false or absurd result.

Here is a simple example. Let us prove the following theorem:

There exists no smallest positive rational number.

Suppose, to the contrary, that there exists a positive rational number r that is smaller than all other rational numbers. By definition, $r = m/n$ for some integers m and n. Consider the number $r/2$. It is certainly rational (since $r/2 = m/2n$, which is a itself ratio of integers) and smaller than r since halving a positive number reduces its value. However, this is a *contradiction*, since we have assumed that r is smaller than *all* other rational numbers. Therefore, the theorem is true.

Here is another example. Suppose there are two categories of people, *knights* and *knaves*. Knights always tell the truth, while knaves always tell a lie. We meet two people, Jack and Jill, but we don't know which category they belong to. They make the following claims:

Jack: "Jill is a knight!"

Jill: "We are of opposite types!"

We deduce their categories as follows. In fact, we will first prove the following "theorem":

Jack is a knave.

Suppose, to the contrary, that Jack is a knight. Since knights tell the truth, Jill is also a knight. Therefore, Jill's statement is also true and Jack is of the opposite type as Jill, i.e., he is a knave. However, this is a *contradiction* since we have assumed Jack is a knight. Therefore, the theorem is true.

Since Jack is a knave, and Jack claims that Jill is a knight, it logically follows that:

Jill is also a knave.

One last example (since proofs by contradiction are important!). This is a very common example given in several textbooks. Let us prove the following theorem:

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

Suppose, to the contrary, that $\sqrt{2}$ is rational. By assumption, we can write $\sqrt{2} = n/d$ in *lowest terms*, i.e., n and d have no common factors. We square both sides and multiply cross terms to get:

$$n^2 = 2d^2$$

Since the right hand side is a multiple of 2, the quantity n^2 is even (and consequently, by the discussion of the last lecture, n is also even.) Writing $n = 2q$ and canceling a factor 2, we get:

$$2q^2 = d^2.$$

This shows that d^2 is *also* a multiple of 2.

In other words, both d and n are even. But this is a *contradiction* since by our assumption above, n and d cannot have any common factors. Therefore, the theorem is true.

Proof by counterexample

Counterexamples are important mathematical constructs. While it is often very difficult to *prove* a mathematical statement, a single counterexample can be used to *disprove* its validity. All we need to show is that the statement is false in at least one situation.For example, suppose we have a statement:

> If n is prime, then n is odd.

Clearly, $n = 2$ is a counterexample. Therefore, the theorem is false.

Often, finding a counterexample can be challenging. For example,

> For every nonnegative integer n, the number $n^2 + n + 41$ is a prime.

Let $P(n)$ be the predicate "$n^2 + n + 41$ is prime". Then, $P(n)$ indeed appears to be true for $n = 0, 1, 2, \ldots$. In fact it is true all the way up to $P(39)$. However, $P(40)$ is false since $40^2 + 40 + 1 = 41 \cdot 41$ which is not a prime number. Therefore, the theorem is false.

Proof by construction

In contrast, proofs by construction are used to prove propositions of the form:

> $\exists x, P(x)$.

To prove such as statement, we only need to demonstrate that the predicate $P(x)$ is true for *one* element in the domain of discourse; if we can do that, then we are done.

For example, suppose we need to prove the theorem:

> There exist positive integers a, b, c, n such that $a^n + b^n = c^n$.

Consider $n = 1$ and $a = 1, b = 2, c = 3$. For this choice of numbers, clearly $a^n + b^n = c^n$ and hence the theorem is true *by construction*.

Equations of the form $a^n + b^n = c^n$ are examples of *Diophantine equations*. In fact, it is known that for any integer $n > 2$, there exist *no* combination of integers a, b, c that satisfy the above equation – this is the celebrated *Fermat's Last Theorem*, a famous problem in mathematics that remained unsolved for over 350 years.

Proof by cases

Sometimes, a mathematical statement needs to be broken down into smaller sub-cases that are more easily proved. Here is an example. Suppose we wish to prove that for any real number x,

$$x + |x - 7| \geq 7.$$

We will consider the following cases.

- Case 1: Assume that $x \geq 7$. Then $x - 7 \geq 0$ and $|x - 7| = x - 7$, so that

$$\begin{aligned} x + |x - 7| &= x + (x - 7) \\ &= 2x - 7 \geq 2(7) - 7 \\ &= 14 - 7 = 7, \end{aligned}$$

i.e., $x + |x - 7| \geq 7$.

- Case 2: Assume that $x < 7$. Then $x - 7 < 0$ and $|x - 7| = -(x - 7) = 7 - x$, so that $x + |x - 7| = x + (7 - x) = 7 \geq 7$, i.e., $x + |x - 7| \geq 7$.

Thus, for all possible cases, we have shown that the theorem is true.

In a proof-by-cases, the first (and most important) thing to do is to identify all possible sub-cases and not miss any. For example, suppose we need to show that for any real numbers x and y,

$$|x + y| \leq |x| + |y|$$

This relation is known as the *triangle inequality*. Here is a sketch of a proof-by-cases. We assume four possible cases:

- both x and y are nonnegative
- x is nonnegative and y is negative.
- x is negative and y is nonnegative.
- both x and y are negative.

In each of these cases, we can write out the left and right hand sides of the triangle unequality and show that the relation indeed holds. We leave this as an easy **exercise.**

Exhaustive proof

This is a special instance of proof-by-cases. If the domain size is small, one can simply check all possible cases by hand and show that the claimed theorem is true.

For example, to prove the statement:

> If n is a positive integer ≤ 5, then $(n + 1)^2 \geq 2^n$,

we can simply enumerate all feasible values of n, i.e., 1 through 5, and manually check that $(n + 1)^2$ exceeds 2^n in each case.

Similar to trivial and vacuous proofs, such proofs are rather rare; typically, one uses this as a smaller step in a larger proof.

Some rules of thumb

That was rather a long list of proof techniques.

Some rules of thumb could be of help:

- For proving an "if-then-", consider using direct proofs, proofs by contraposition, or contradiction.
- For proving an "A if and only if B", think of proving both "A implies B" and "B implies A".
- For proving "There exists - such that -", think of a constructive proof.
- For disproving a statement, consider showing a counterexample.
- Use rules of inference wherever you can.

Of course, these are not guaranteed to work, and some techniques might be easier to use in certain cases than others. As always, there is only way to get better at constructing proofs: practice!

A note on writing good proofs

As mentioned earlier: writing out a proof is similar in spirit to writing out a large and complex piece of code.

Similar to writing good code, a good proof must not only be logically correct and precise, but also be clearly written. All variables must be defined clearly; all assumptions must be either stated or justified; and all rules must be correctly applied.

A well-written proof is more likely to be correct and logical bugs will be easier to fix. On the other hand, a poorly written proof (even if correct) loses much of its value.

Here are some tips for writing good proofs:

- State your proof technique in the beginning. It is much easier to identify the flow of a proof if you (and the reader) knows what technique is being used. Begin with something like "We will prove this statement by contraposition. . . . ".
- Keep the flow linear, i.e., one statement should lead in to the next and so on.
- Avoid using arcane mathematical notation. Use ordinary language.
- If a proof is long, break it down into smaller, digestible pieces.
- Revise and simplify your proof once you are done.
- Revise again! Check for common pitfalls, counterexamples, and possible missing cases.
- Conclude. Bring the discussion back to where it started and explain why the original statement is true. Many proofs end with the letters "Q.E.D.", which is an abbreviation of *quod erat demonstrandum*, a Latin phrase that means "which is what we set out to show".

Chapter 9

Sets

Recall the notion of a *data type*, essential for specifying the kinds of objects that software programs can use and manipulate. Analogous to this notion, we will begin the study of (somewhat) more general, *mathematical* data types. In particular, we will dig deeper into three types: sets, functions, and relations.

Set theory basics

Sets are simple to understand, but hard to define precisely.

Informally, a set is a bunch of objects; these objects are called the *elements*, or *members* of the set.

The definition of "object" is deliberately general, and could be just about anything: numbers, letters, points on a plane, strings, names of CPRE students, football teams, and so on. In fact, the elements of a set can even be *other sets*. Our convention is that whenever we list the elements of a set, we do so within curly braces {}.

Some examples of sets:

- The set of letters in the English alphabet: $A = \{a, b, c, \ldots, z\}$
- The set of even integers strictly between 5 and 10: $E = \{6, 8\}$
- The set of living ex-Presidents: $P = \{\text{Carter, Bush41, Clinton, Bush43}\}$
- A set of sets: $\{\{a, b\}, \{b, c\}, \{a, c\}\}$

We use the symbol "$\in$" to denote set membership; i.e., if x is a member of some set S, then we write it concisely as:

$$x \in S$$

Similarly, to say that x is not a member of some set S, we write it as:

$$x \notin S$$

Sets are typically described by identifying some specific property that is satisfied by all its elements. Such sets can be denoted using *set builder notation* by stating the property as a

predicate. For instance, instead of listing out all the alphabets in the first example, we can alternatively specify A as:

$A = \{x | x \text{ is a letter in the English alphabet}\}$

Properties of sets

There are two main properties of sets to keep in mind:

1. The *order* of specifying the elements in a set is not important. For example, the set $\{6, 8\}$ is identical to the set $\{8, 6\}$, just written down in a different way. (Later, we will talk about *sequences*, where the order or elements will be vitally important.)
2. *Repetition* of elements is not important: there is no notion of an element appearing more than once in a set. For example, the set $\{6, 8\}$ is the same as the set $\{6, 6, 8, 6, 8\}$. By convention, we will remove all duplicates while enumerating the contents of a set.

Some important sets

In mathematical proofs, the following sets often come up over and over again:

- The set of integers: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- The set of nonnegative integers: $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$
- The set of positive integers: $\mathbb{Z}^+ = \{1, 2, \ldots\}$

Using predicates, we can also write $\mathbb{Z}^+$ as $\mathbb{Z}^+ = \{x | (x \in Z) \wedge (x > 0)\}$

- The set of real numbers: $\mathbb{R} = \{x | x \text{ is real}\}$
- The set of positive real numbers: $\mathbb{R}^+ = \{x | x \text{ is real} \wedge x > 0\}$
- The set of rational numbers: $\mathbb{Q} = \{x | x = \frac{a}{b}, a, b \in \mathbb{Z}, b \neq 0\}$
- The empty set (or the *null* set) $\phi = \{\ \}$

Sometimes we talk about:

- The *universal* set, which consists of all elements in a given domain of discourse (or universe). This is not an *a priori* fixed set like the above definitions, but depends on what we are talking about.

Equality of sets

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

In symbols: let A and B be sets. Stating that $A = B$ is logically equivalent to the proposition:

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

This proposition can be used as an algorithm for checking the equality of two given sets via enumeration: cycle through each element of the first set, and check if it belongs to the second; cycle through each element in the second set and check that this belongs to the first.

For example, one can check (via exhaustive enumeration) that the set $E = \{4, 6, 7\}$ is equal to $F = \{7, 6, 4\}$.

This proposition is the first of a collection of axioms that underlie set theory, and is called the *principle of extension*. This collection of axioms is called "ZFC" (named after mathematicians Zermelo and Frankel), and can be used to construct most of the existing concepts in modern math that we take for granted. But that's a much deeper story, best deferred to a more advanced course in formal logic.

Subsets

A set S is called a *subset* of another set T if **every** element of S also belongs to T.

If S is a subset of T, we use the notation $S \subseteq T$. Think of $\subseteq$ as pointing to a smaller set, just as how the symbol $\leq$ points to a smaller number.

In symbols: let A and B be sets. Then, stating that $A \subseteq B$ is logically equivalent to the proposition:

$$\forall x(x \in A \implies x \in B)$$

Notice the (subtle) difference from the definition of set equality defined above. To check that some set A is a subset of some bigger set B, cycle through each element of A and verify that it is a member of B.

As an example: if $U = \{2, 4, 6\}$, $V = \{1, 2, 3, 4, 5, 6\}$ and $W = \{2, 4, 5\}$, we observe that the following are true:

$$U \subseteq V$$

$$W \subseteq V$$

On the other hand, consider:

$$U \subseteq W$$

This statement is **false** since $6 \in U$ but $6 \notin W$. That is, U is not a subset of W since there at least one element in U that is not an element in W. Symbolically, we denote this as $U \nsubseteq W$.

Some properties of subsets

0. We say that A is a *proper* subset of B if $A \subseteq B$ but $A \neq B$, i.e., there is at least one element of B that is not in A. We denote this by the symbol $\subset$.

1. By definition, any set is a subset of itself, i.e., $A \subseteq A$.

2. Also, by definition, the null set ϕ is a subset of any arbitrary set S, i.e., $\phi \in S$. This can be proved as follows. Invoking the definition of $\subseteq$, we need to show that for any arbitrary set S:

$\forall x(x \in \phi \implies x \in S)$

3. However, $x \in \phi$ is *false* for any x since ϕ is empty. Therefore, the implication_is true (recall the property of $\implies$). This is an example of a *vacuous proof.*

4. Note that $\subseteq$ is **distinct** from $\in$. This is a common type error that people make often. Writing down $1 \in \{1, 2, 3\}$ is *not* the same as $\{1\} \subseteq \{1, 2, 3\}$. In the former, the number 1 denotes an *element* of the *set* $\{1, 2, 3\}$, while in the latter, the set $\{1\}$ denotes a *subset* of the *set* $\{1, 2, 3\}$. Watch out for the curly braces!

5. The *power* set is the set of all subsets of some given set S. For example, if $S = \{6, 8\}$, then the power set of S, denoted by $\mathbb{P}(S)$ is the set:

   $\{\{6\}, \{8\}, \{6, 8\}, \phi\}$

Two things to note here. First, each of the elements of $\mathbb{P}(S)$ is itself a set (and therefore is written with curly braces). Next, by the argument we made above, both ϕ and S itself are subsets of S, and therefore are included in the power set.

Let's do an **exercise.** Can you identify which two of the following statements are incorrect?

- $3 \in \{1, 2, 3\}$
- $\{2\} \in \{1, 2, 3\}$
- $2 \subseteq \{1, 2, 3\}$
- $\{3\} \subset \{1, 2, 3\}$
- $\{1\} \subseteq \{1, 2, 3\}$
- $\{3\} \in \{\{2\}, \{3\}\}$

The incorrect ones are the second and third statements. There is a type error in each of these statements; convince yourself why that is the case! Also, the last one is correct: the right hand side is a set of sets; therefore *its* elements are the sets $\{2\}$ and $\{3\}$ and it is accurate to claim that $\{3\}$ is a member of $\{\{2\}, \{3\}\}$.

Finally: If two sets A and B are equal then, $A \subseteq B$, and also, $B \subseteq A$. In fact, the reverse is also true:

If $A \subseteq B$ and $B \subseteq A$, then $A = B$.

This fact is a different way to prove that two sets are equal. We will use this in later lectures.

Cardinality of sets

If a set contains finitely many elements, then it is called a finite set. The number of elements in a finite set is called its *size*, or *cardinality*. If a set is not finite, then its cardinality is not a finite number and the set is said to be *infinite*.

For example, the cardinality of the set of letters in the English alphabet is 26, and the cardinality of the set of living ex-Presidents is 4. The set of real numbers $\mathbb{R}$ contains infinitely many elements (since there are an infinity of real numbers), and hence $\mathbb{R}$ is infinite. By convention, the empty set is said to have cardinality 0.

A useful factoid: if a finite set S has cardinality n, then its power set $\mathbb{P}(S)$ has cardinality 2^n. We will prove this formally by induction, but as an **exercise**, convince yourself that this is the case using the set $S = \{\text{red, blue, green}\}$ as an example.

Chapter 10

Set Operations

We have introduced the notion of a set, and some properties of sets (elements, equality, cardinality, subsets).

We now consider some basic set operations. Specifically, given two or more sets, we will see how to manipulate their elements in non-trivial ways using unions, intersections, complements, products, and so on.

Basic operations on sets

In what follows, let A and B be two generic sets, and U denote the universal set. Let us start with three familiar operations:

- *Union* : The set of elements that belong to either A or B:

$$A \cup B = \{x \mid (x \in A) \vee (x \in B)\}$$

- *Intersection* : The set of elements that are common to both sets:

$$A \cap B = \{x \mid (x \in A) \wedge (x \in B)\}$$

Two sets are called *disjoint* iff their intersection is empty, i.e., $A \cap B = \phi$.

- *Complement* : The set of all elements (in the universal set) that do *not* belong to a given set:

$$\overline{A} = \{x \mid x \notin A\}$$

Sometimes the symbol used for set complement is superscript-c, i.e, A^c.

You have probably seen/heard of these operations before in an earlier course. The typical way to illustrate these operations is via Venn Diagrams.

Here is an example. Suppose the domain of discourse under consideration is the set of positive integers up to 10, i.e., $U = \{1, 2, 3, 4, \ldots, 10\}$. Consider sets $A = \{2, 3, 5, 7\}$ and $B = \{2, 4, 5\}$. Then, it is easy to see that:

- $A \cup B = \{2, 3, 4, 5, 7\}$
- $A \cap B = \{2, 5\}$
- $\overline{A} = \{1, 6, 8, 9, 10\}$

A property (easily proved) of unions and intersections is that they are *commutative*; the order in which you specify the constituent sets doesn't matter. So, $A \cup B$ is the same as $B \cup A$, and $A \cap B$ is the same as $B \cap A$. We will discuss commutativity and several other properties below.

The above definitions of union and intersection can be extended to any number of sets:

$$S_1 \cup S_2 \ldots \cup S_n = \{x \mid (x \in S_1) \vee (x \in S_2) \ldots \vee (x \in S_n)\}$$

$$S_1 \cap S_2 \ldots \cap S_n = \{x \mid (x \in S_1) \wedge (x \in S_2) \ldots \wedge (x \in S_n)\}$$

There are two operations that are somewhat less commonly encountered in introductory courses:

- *Set difference*: The set of elements that belong to one set but not the second:

$$A - B = \{x \mid (x \in A) \wedge (x \notin B)\}$$

In the above example, $A - B$ is the set $\{3, 7\}$ since 3 and 7 are members of A but not B. Note that this is **different** from $B - A$, which is the set $\{4\}$. In that sense, set difference is *not commutative*.

One can combine the definitions of complement and intersection to arrive at the identity:

$$A - B = A \cap \overline{B}$$

Set differences satisfy the property that $A - B$ and $B - A$ are disjoint. As an **exercise**, try proving this by contradiction.

- *Symmetric difference*: The set of elements that belong to either one set of the other, but not both:

$$A \oplus B = \{x \mid (x \in A) \oplus (x \in B)\}$$

In contrast to ordinary set difference, symmetric difference *is* commutative. Try proving the following identity:

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

Set operations and cardinality relations

Let $|A|$ denote the cardinality of a set A. There are some interesting relationships between the cardinality of a pair of sets, their unions and intersections, etc.

First, we easily see that the following relationships are true: $|A \cap B| \leq |A|$ and $|A \cap B| \leq |B|$. This is because intersecting a set with some other set can never increase its size. Moreover, $|A \cup B| \geq |A|$ and $|A \cup B| \geq |B|$; taking the union of a set with another set cannot decrease its size.

Here is a more non-trivial relationship. In the above example involving sets of numbers up to 10, we observe that $|A| = 4, |B| = 3$. Moreover, $|A \cup B| = 5$ and $|A \cap B| = 2$.

More generally, we have the following theorem:

For any pair of finite sets A and B, $|A| + |B| \geq |A \cup B|$.

We will give a full, rigorous proof of this theorem later. In fact, we will prove a more general principle. Observe that $|A \cup B| = 5$ and $|A \cap B| = 2$, which rather conveniently adds up to the sum of the sizes of $|A|$ and $|B|$.

This is an instance of a more general result called the *principle of inclusion-exclusion* (PIE). In its simplest form, this principle can be stated as follows:

For any pair of finite sets A and B, $|A| + |B| = |A \cup B| + |A \cap B|$.

If one takes the PIE for granted, then the theorem stated above follows quite easily, since $|A \cap B|$ cannot be negative (and therefore, $|A| + |B|$ must be at least as big as $A \cup B$.)

Cartesian products

A somewhat different type of set operation is the *Cartesian product*. This is named after Descartes, the famous French mathematician from the 17th century. Here, the word "product" is different from what we typically call a product between numbers, although we use the same symbol $\times$.

To define Cartesian products, we first introduce the concept of an *ordered pair*. An ordered pair, or a *tuple*, is a collection of two objects (a_1, a_2) such that a_1 is designated as the first element and a_2 is designated as the second element.

Note that this is different from the *set* $\{a_1, a_2\}$. Ordering of elements does not matter while talking about sets, i.e., $\{a_1, a_2\} = \{a_2, a_1\}$. However, order matters while talking about tuples; $(a_1, a_2) \neq (a_2, a_1)$. We will use the regular parentheses $(\cdot, \cdot)$ to denote tuples.

The notion of ordered pairs can be generalized to more than 2 elements. An ordered n-*tuple is simply the ordered collection of elements $(a_1, a_2, a_3, \ldots, a_n)$.

We now introduce Cartesian products. Let A and B be two sets. Then, the Cartesian product of A and B is the set of all ordered pairs (a, b) such that a belongs to A and b belongs to B, i.e,.

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

For example, if S denotes the set of ISU students, and C denotes the set of computer engineering courses, then:

$$S \times C = \{(s, c) \mid s \in S, c \in C\}$$

denotes the set of all possible enrollments of students in computer engineering courses.

Three quick points. First, observe that taking Cartesian products of sets is *not* a commutative operation; $A \times B$ is very different from $B \times A$ (unless $A = B$).

Second, the number of possible ordered pairs (a, b), by a simple counting argument is given by $|A|$ multiplied by $|B|$. In other words, the cardinality of the Cartesian product of two sets is the product of their cardinalities.

Lastly, observe that we can generalize Cartesian products to more than 2 sets using n-tuples. The Cartesian product of n sets, $A_1 \times A_2 \times \ldots \times A_n$, can be defined as the set of all n-tuples $(a_1, a_2, \ldots, a_n)$ such that $a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n$.

The power of Cartesian products is that they can be used to construct concise descriptions of large, complex sets. For example, suppose we are interested in the set of all possible license plate numbers in Iowa. For simplicity, let us assume that license plate numbers in Iowa consist of 3 letters of the alphabet (in caps) followed by 3 digits. We can concisely describe this set as follows. Let A be the set of letters in the alphabet (in caps) and D be the set of digits. Then, the set of possible license plate numbers can be written as:

$$A \times A \times A \times D \times D \times D,$$

or $A^3 \times D^3$ for short.

On the other hand, suppose that license plates in Texas consist of 6 alpha-numeric symbols in any arbitrary order. Therefore, the set of license plate numbers in Texas can be denoted as:

$$(A \cup L) \times (A \cup L) \times (A \cup L) \times (A \cup L) \times (A \cup L) \times (A \cup L),$$

or $(A \cup L)^6$ for short.

Properties of set operations

Set operations obey a list of properties (laws). We have already discussed the commutativity property above. It is useful to know and understand each of these properties; many of them can be visualized by drawing a quick Venn diagram and verifying that they are correct. When in doubt, try instantiating each of the following identities with any example sets A, B, C and ensure that they make sense. An alternate way to visualize these identities is to draw the Venn diagram and shade the corresponding regions (such a procedure is informally called a "proof by picture".)

In all expressions below, U denotes the universal set and ϕ denotes the empty set. The first five laws only involve a single set A.

- Identity laws:
$$A \cup \phi = A, \quad A \cap U = A$$
- Domination laws:
$$A \cup U = A, \quad A \cap \phi = \phi$$
- Idempotent laws:
$$A \cup A = A, \quad A \cap A = A$$

- Complement laws:
$$A \cup \overline{A} = U, \quad A \cap \overline{A} = \phi$$

- Involution (double complement):
$$A = \overline{\overline{A}}$$

Most of the above laws are more or less taken for granted. Also, if you stare at these relations carefully, you will observe that there is a curious resemblance to the more familiar relations of Boolean logic that we have seen earlier: replace A with a propositional variable; replace $\cup$ and $\cap$ with $\wedge$ and $\vee$; replace complements with $\neg$; and we get almost exactly the same identities as above.

- Commutative laws:
$$A \cup B = B \cup A, \quad A \cap B = B \cap A$$

- Absorption laws:
$$A \cup (A \cap B) = A, \quad A \cap (A \cup B) = A$$

- Associative laws:
$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

- Distributive laws:
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

- De Morgan's laws
$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Again, these identities look suspiciously like Boolean logic formulas!

In fact, one way to interpret (and even prove) the above identities is via special truth tables known as *set membership tables*. We consider all possible combinations that an element can (or cannot) belong to, and verify that the same combinations leads to both sides of the identity. For example, to prove the first Absorption law, suppose we assign "T" if a fixed element x belongs to a given set, and "F" otherwise. Then, the following table lists all possible membership configurations of x:

A	B	$A \cap B$	$A \cup (A \cap B)$
T	T	T	T
T	F	F	T
F	T	F	F
F	F	F	F

Since the last column, corresponding to $A \cup (A \cap B)$, has an identical set membership value

as the first column, corresponding to A, the two sets are equal. Indeed, this can be viewed as a formal proof of the first Absorption Law. As an **exercise**, try constructing proofs of all the other laws using a similar procedure.

Chapter 11

Functions

In several applications, we want to assign, to each element of a given set, a specific element of a second set. Such assignments are instances of *functions*. The notion of a "function" is fundamental to all of math and computer science.

Definitions

Let X and Y be nonempty sets. A function (also called a *mapping* or *transformation*) from X to Y maps an element in X to exactly one (unique) element in Y. We denote this mapping using the notation $f : X \to Y$. Sometimes, x is called the *argument* of the function, and y is called the *value*.

The set X is called the *domain* of f, while the set Y is called the *co-domain* of f. For short, we use the notation $\text{Dom}(f)$ and $\text{Co-dom}(f)$.

The unique element in Y that $x \in X$ gets mapped to is called the *image* of x under f. If $f(x) = y$, then x is called a *pre-image* of y.

It is important to note that while *not* every element in Y necessarily gets a pre-image. (In other words, there could be elements in Y that do not have any corresponding x.) The subset of Y that consist of all elements y that are images is called the *range* of f, denoted by $\text{Ran}(f)$. In set-builder notation, we can express this as:

$$\text{Ran}(f) = \{y \in Y \mid \exists x \in X,\ f(x) = y\}.$$

For example, if at the end of the class, each student in CPRE 310 will be assigned a letter grade. Grades are assigned to each student. Moreover, this grade is unique (i.e., students get exactly one grade in the end.). Therefore, we can model this assignment as a function f.

Let's use a simpler example. Suppose there are 5 people in a class - Ava, Bob, Chuck, Don, and Emma. Their grade assignments (in order) are B, B, A, C, A.

We model this as follows. Define $X = \{\text{Ava}, \text{Bob}, \text{Chuck}, \text{Don}, \text{Emma}\}$ be the set of students in the class, $Y = \{A, B, C, D, F\}$ be the set of letter grades. Let f denote the grade assignment function. Then, we have:

- $f(\text{Ava}) = B$
- $f(\text{Bob}) = B$
- $f(\text{Chuck}) = A$
- $f(\text{Don}) = C$
- $f(\text{Emma}) = A$

Moreover, $\text{Dom}(f)$ is X, $\text{Co-dom}(f)$ is Y, and $\text{Ran}(f) = \{A, B, C\}$ is the set of all grades that the students in the class end up getting. (If everyone got an A, then $\text{Ran}(f)$ would be $\{A\}$.)

Inverse image

Consider some function f, and let y be some element in $\text{Ran}(f)$. The *inverse image* of y, denoted as $f^{-1}(y)$, is the set of all elements $x \in X$ such that $f(x) = y$. As opposed to $f(x)$, which is an *element* of Y, keep in mind that $f^{-1}(y)$ is a *subset* of elements of X. In set-builder notation, we have:

$$f^{-1}(y) = \{x \in X \mid f(x) = y\}$$

In the above example with grades, we have:

- $f^{-1}(A) = \{\text{Chuck}, \text{Emma}\}$
- $f^{-1}(B) = \{\text{Ava}, \text{Bob}\}$
- $f^{-1}(C) = \{\text{Don}\}$

Note that $f^{-1}(D)$ or $f^{-1}(E)$ are **not defined** since D and E do not belong to $\text{Ran}(f)$.

Some examples

Let us do a few more examples.

1. Let $X = Y = \mathbb{R}$. Suppose that $f(x) = x + 1$. Then, $\text{Dom}(f) = \mathbb{R}$, $\text{Co-dom}(f) = \mathbb{R}$ (since the function maps real numbers to real numbers), and $\text{Ran}(f) = \mathbb{R}$ as well, since every real number y is the image of some x under f.
2. On the other hand, $x^2 + y^2 = 1$ is **not** a function the way we have defined it. This is because if we solve for x, we can see that $x = \pm\sqrt{1 - y^2}$, i.e., x is assigned two possible values. This is not allowed under our definition above since functions map elements to *uniquely* defined values.
3. Let $X = Y = \mathbb{R}$. Suppose that $f(x) = 0$. Then, $\text{Dom}(f) = \mathbb{R}$, $\text{Co-dom}(f) = \mathbb{R}$, and $\text{Ran}(f) = 0$.

Chapter 12

Types of Functions

In the last lecture, we gave a definition of a *function*, and established various notions such as its domain, co-domain, and range. We also encountered ideas such as the image, pre-image, and inverse image.

First, a convention: we will always assume that any function f that we discuss is *everywhere well-defined*, i.e., each element in the domain X will have a unique image under f. In terms of an example, under the "grade-assignment function" that we defined previously (where the mapping is from the set of CPRE 310 students to the set of letter grades), the everywhere-well-defined property implies that *every* student in 310 gets mapped to a letter grade.

Injective function

Injective functions are also known as *one-to-one* functions. Injective functions are functions where distinct elements in the domain get mapped to distinct elements in the co-domain. Formally, the injective property can be captured in terms of predicate logic:

$$\forall x_1, x_2 \in X,\ f(x_1) = f(x_2) \implies x_1 = x_2$$

The easiest way to imagine a one-to-one function is by drawing arrows from the domain X to the co-domain Y. If each element in the co-domain has **at most** 1 "incoming" arrow, then the function is injective.

As a simple example, if $X = 1, 2, 3, 4$ and $Y = A, B, C, D$, consider the function f defined as:

$$f(1) = A, f(2) = D, f(3) = C, f(4) = B$$

This function would be a one-to-one function. On the other hand, if the function were defined as:

$$f(1) = f(2) = f(3) = f(4) = B$$

then the function would *not* be a one-to-one function.

Here are some other examples. The function $f : \mathbb{R} \to \mathbb{R}$ defined as $f(x) = 2x + 3$ is injective. We prove this by contradiction. Suppose, to the contrary, that the function is not injective, i.e., there exist two distinct numbers x_1, x_2 such that $f(x_1) = f(x_2)$. By definition of f, we have:

$$2x_1 + 3 = 2x_2 + 3$$

Solving for x_1, we get $x_1 = x_2$, which contradicts the assumption that x_1 and x_2 are distinct. Therefore, the function is injective.

On the other hand, the function $f(x) = x^2$ is not injective. A direct proof would simply follow by observing that for any $x \neq 0$, $f(x) = f(-x) = x^2$.

In general, when discussing functions defined over real numbers, try to find out if the function is *strictly* increasing or *strictly* decreasing. (For instance, linear functions that are not constant are either increasing or decreasing; so are exponential functions; so are log functions.) In all these cases, the function can be proved to be one-to-one.

Surjective function

Surjective functions are also known as *onto* functions. Surjective functions are functions where every element in the co-domain is the image of some element in the domain. Formally, the surjective property is defined as:

$$\forall y \in Y,\ \exists x \in X,\ f(x) = y.$$

Again, the easiest way to imagine an onto function is by drawing arrows from the domain X to the co-domain Y. If each element in the co-domain has **at least** 1 incoming arrow, then the function is surjective.

In the example above where $X = 1, 2, 3, 4$ and $Y = A, B, C, D$, the function f is surjective (in addition to being injective.) Observe that each element in Y has some x that gets mapped to it.

The function $f(x) = 2x + 3$ is surjective if the domain of f is specified as $\mathbb{R}$. This is because every real number (say, y) can always be written as $y = 2x + 3$ for *some* real number x.

However, the function $f(x) = 2x + 3$ is *not* surjective if the domain of f is specified as the set of integers $\mathbb{Z}$. For example, $y = 2$ cannot be written as $2 = 2x + 3$ for *integer* x.

The function $f(x) = x^2$ is not surjective if the domain is $\mathbb{R}$ *or* $\mathbb{Z}$. Try proving this! For now, we leave this as an **exercise**.

Bijective function

Bijective functions are also known as *one-to-one correspondence* functions. Bijective functions are functions that are both one-to-one as well as onto.

Back to our mental picture of arrows from X to Y. A bijective function will have: * one arrow out of every element in X (since it is a function) * **exactly** arrow into **every** element in Y.

Examples: the linear function $f(x) = 2x + 3$, defined over the real numbers as the domain, is bijective; we proved above that it is both one-to-one as well as onto. Any linear function of the form $f(x) = ax + b$ will be a one-to-one correspondence in general, unless $a = 0$ (in which case the function f is a constant/flat function.)

A simpler example is the *identity* function $f(x) = x$. (Here, the domain could be any set A.)

On the other hand, $f(x) = x^2$ is *not* a bijection if the domain of f is $\mathbb{R}$. First of all, f is not one-to-one since $f(1) = f(-1)$. Moreover, f is not onto since negative real numbers have no pre-image under f.

Some useful "Computer Science" functions

Let us now discuss some common functions that often arise in CPRE/SE applications.

1. Consider any set S that is a subset of a given universal set U. The *characteristic* function f with respect to S is defined as:

$$f(a) = \begin{cases} 1, & \text{for } a \in S \\ 0, & \text{for } a \notin S \end{cases}$$

   The characteristic function is onto if S is a proper subset of U, but not one-to-one.

2. The *ceiling* function $f : \mathbb{R} \to \mathbb{Z}$, denoted by $f(x) = \lceil x \rceil$ is the smallest integer that is greater than or equal to x.

3. The *floor* function $f : \mathbb{R} \to \mathbb{Z}$, denoted by $f(x) = \lfloor x \rfloor$ is the largest integer that is smaller than or equal to x. Neither the ceiling nor the floor functions are one-to-one, but both are onto.

4. The *modulo* function $f_p : \mathbb{Z} \to \mathbb{Z}$, denoted by $f_p(x) = x \mod p$, is the remainder when x is divided by p. This function is neither one-to-one nor onto.

Rules of thumb

It is often important to formally prove whether a function f is injective/surjective/bijective. Here are some rules of thumb that can be followed for constructing such proofs:

- To prove that f is one-to-one, try doing a proof-by-contradiction. Assume that $x_1 \neq x_2$ but $f(x_1) = f(x_2)$. Somehow deduce that x_1 and x_2 must be equal, thus leading to a contradiction.
- To prove that f is onto, try doing a direct proof. Assume some generic y in the co-domain and proving that $y = f(x)$ for some x in the domain.

- To disprove that f is one-to-one, try doing a proof-by-counterexample: find some pair x_1, x_2 such that $f(x_1) = f(x_2)$. To disprove that f is onto, also try a counterexample: find y that has no inverse image in X.

Chapter 13

More on functions; sequences

We have heard about injective, surjective, and bijective functions. Let us now use these ideas to develop two important ways to transform functions; how to compute function *inverses*, and how to do *compositions* of two or more functions.

Inverse of a function

Let $f : X \to Y$ be a *bijective* function. That is, there is a one-to-one correspondence between the elements of X and Y. Then, the *inverse* of f, denoted by $f^{-1} : Y \to X$ is the function defined as follows:

$$f(x) = y \iff f^{-1}(y) = x.$$

In words, f^{-1} "undoes" the action of f, and sends each element of Y back to their (unique) pre-image in X.

For example, let $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x + 7$. Then, its inverse function is defined as $f^{-1}(x) = x - 7$. Going back to our understanding of functions in terms of arrows from X to Y: any inverse function f^{-1} can be constructed by simply reversing the direction of each arrow in f.

Important note: inverse functions exist and are well-defined only if the function is one-to-one as well as onto (such functions are said to be *invertible*.) Here is an example where this is not the case. The function $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x^2$ is *not* a bijection; for instance, we can easily check that $f(3) = f(-3) = 9$ and the function is not one-to-one. Therefore, it does not have a well-defined inverse.

Another important note: keep an eye out for the domain and co-domain when discussing function inverses. For example, if $f : \mathbb{R}^+ \to \mathbb{R}^+$ (positive reals to positive reals) and $f(x) = x^2$, then f *is* bijective (as an **exercise**, check that it is both one-to-one and onto). Here, the inverse of f is given by $f^{-1}(x) = \sqrt{x}$.

Function compositions

Let $f : A \to B$ and $g : B \to C$ be functions. Then the composition of f and g, denoted by $g \circ f$, is defined for any $x \in A$ as:

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

For example, let all domains/co-domains be $\mathbb{R}$, and define $f(x) = 2x + 1$ and $g(x) = 5x + 7$. Then,

$$g \circ f(x) = g(2x + 1) = 5(2x + 1) + 7 = 10x + 12.$$

One could also define $(f \circ g)(x)$ the same way, except with the order reversed:

$$f \circ g(x) = f(5x + 7) = 2(5x + 7) + 1 = 10x + 15,$$

which is a *different* function than $g \circ f(x)$. In general, function composition is not commutative, i.e., $f \circ g \neq g \circ f$.

A natural adaptation of function compositions is the notion of an *iterated* function. If $f : X \to X$, then we can compose f with itself, i.e., $f^2 = f \circ f$. Similarly, we can define $f^3 = f \circ f^2 = f \circ f \circ f$, and so on until we get $f^n = f \circ \overset{n \text{ times}}{\cdots} \circ f$. For example, if $f(x) = 2x$, then $f^2(x) = 2(2x) = 4x$, $f^3(x) = 2(4x) = 8x$, etc. A more concise way to represent the function iteration is by using the *recursive* definition:

$$f^1 = f, \quad f^n = f \circ f^{n-1}.$$

More on recursive definitions shortly.

One other point on function compositions. Let us define the (trivial) *identity* function on any set A as $i_A(x) = x$ for all x. Now, consider any invertible function $f : A \to B$ (i.e., $f^{-1} : B \to A$). It is not too hard to prove that

$$f^{-1} \circ f = i_A$$

and

$$f \circ f^{-1} = i_B$$

In words, the function composed with its inverse gives the identity function defined on the domain that you started with. So if $f(x) = x^2$ and $g(x) = \sqrt{x}$ (where the domain/co-domain are the positive reals), then $(f \circ g)(x) = (g \circ f)(x) = x$.

Sequences

Recall the definition of ordered pairs and n-tuples, which are collections of elements specified in a fixed order. Both are examples of *sequences*. More generally, a sequence is defined as any ordered list of elements. This is in contrast with a set (where there is no concept of ordering of elements.) A sequence can be either finite or infinite, depending on how many elements it contains.

Formally, a sequence is defined as follows. Let $\mathbb{Z}$ be the integers. Take any subset $A \subseteq \mathbb{Z}$ as the domain, and a given set S as the co-domain. Then, a sequence is any function $f : A \to S$.

Since they are functions defined over integers, we typically denote sequences using the symbol (a_n).

Here are some examples:

0. The sequence of positive integers between 1 and 5: $1, 2, 3, 4, 5$
1. The sequence of nonnegative integers: $0, 1, 2, \ldots, n, \ldots$
2. The sequence of even integers: $\ldots, -2, 0, 2, 4, \ldots$
3. The sequence of reciprocals of integers: $1, \frac{1}{2}, \frac{1}{3}, \ldots$
4. The sequence of powers of 2: $1, 2, 2^2, 2^3, \ldots, 2^n, \ldots$

The above are all examples of *numerical* sequences, where the co-domains are all numbers. But keep in mind that one can define sequences of any type, depending on the choice of co-domain. For example, the list of Super Bowl winners from 1967 to present (in chronological order) is an example of a sequence.

The convention in computer science and engineering (for both finite and infinite sequences) is to start numbering the sequence from zero, i.e., the elements are a_0, a_1, and so on.

A special type of sequence is a *string*. Strings are finite sequences where the co-domain S consists of a set of symbols. For example, if $S = a, b, c$, then *aabbaaaccab* and *aaa* are examples of strings. The base set of symbols depends on the application: it could refer to bits (0 or 1), or numerals, or an specific alphabet, or generic ASCII characters, or pretty much any other symbolic set.

Aside: You are probably somewhat familiar with strings; they are a fundamental data type in most (all?) programming languages. The *length* of a string is the number of elements in the sequence of characters. The empty string (or null string) is a hypothetical string with zero characters, and is denoted using the symbol λ.

Recursive construction of sequences

Since sequences can be very long (or even infinite), we sometimes need to come up with a concise way to construct and represent them. A powerful method is via *recursive constructions*, or recursions for short.

Let us explain this using an example. Consider the infinite sequence consisting of all integers greater than or equal to 4:

$$4, 5, 6, 7, 8, \ldots,$$

Instead of manually listing out the (infinite) set of integers, one can instead concisely define the above sequence as follows:

$$\begin{aligned} &\text{(base case) } a_0 = 4 \\ &\text{(recursion) } a_n = a_{n-1} + 1 \ \forall n \ \geq 1. \end{aligned}$$

Observe the structure of the above definition. There is a *base case* (denoting the initial value of the sequence) and a *recursion* that defines how the next value depends on the previous value.

On the other hand, consider:

$$4, 7, 10, 13, 16, 19, \ldots.$$

This may appear more complicated looking than the previous sequence, but not really! The recursive definition is (almost) identical to the one above:

$$\begin{aligned} &\text{(base case) } a_0 = 4 \\ &\text{(recursion) } a_n = a_{n-1} + 3 \ \forall n \ \geq 1. \end{aligned}$$

The base remains the same, while the step size changes. In general, any sequence of the form:

$$b, b+d, b+2d, b+3d, \ldots, b+(n-1)d, b+nd, \ldots$$

can be written in an identical recursive definition as above:

$$\begin{aligned} a_0 &= b, \\ a_n &= a_{n-1} + d. \end{aligned}$$

Such sequences have a special name: they are called *arithmetic series*.

Likewise, consider:

$$1, 2, 4, 8, 16, 32, 64, \ldots.$$

The above sequence has the following recursive definition:

$$\begin{aligned} &\text{(base case) } a_0 = 1 \\ &\text{(recursion) } a_n = 2a_{n-1} \ \forall n \ \geq 1. \end{aligned}$$

In general, any sequence of the form:

$$b, br, br^2, br^3, \ldots, br^{n-1}, br^n, \ldots$$

has the recursive definition:

$$\begin{aligned} a_0 &= b, \\ a_n &= ra_{n-1}, \end{aligned}$$

and are called *geometric series*.

Here is a slightly more realistic application. Let's say a person starts off with \$500, and deposits \$100 in a savings account every month. The annual interest rate is 6% (spread out evenly over 12 months and compounded monthly). In other words, the savings appreciates by 0.5% at the end of each month. How much money is in the savings account at the end of each month?

The reasoning is as follows: at the end of each month, the previous amount gets multiplied by 1.005, and additionally incremented by 100. Therefore, this can be modeled using recursion. Let (a_n) be the sequence that denotes the amount of savings at the end of the n^{th} month. This sequence follows the following intuitive recursion (all numbers below in dollars):

$$\begin{aligned} a_0 &= 500, \\ a_n &= 1.005 * a_{n-1} + 100. \end{aligned}$$

Chapter 14

Recursion and Summations

We have defined what recursion is, and seen how to model sequences using recursive constructions. Now we will ask the reverse question; given a recursion, how can we *efficiently* figure out a closed form expression or formula that will reconstruct the elements of a sequence. Later in the course, we will re-derive these expression more formally using mathematical induction, but for now we will restrict ourselves to using some amount of guesswork. Fortunately, we can do the guesswork in a couple of different ways.

Recursions and closed form expressions

The simplest example is the arithmetic progression. Recall that an arithmetic progression is any sequence of numbers of the form $(b, b+d, b+2d, b+3d, \ldots)$. Recall that we modeled this using the recursion:

$$a_0 = b,$$
$$a_n = a_{n-1} + d.$$

How do we compute the n^th element of this sequence? The first way is to start at $a_0 = b$, obtain $a_1 = b + d$, $a_2 = a_1 + d = b + 2d$ and so on, until we apply the recursion n times. This is called *forward substitution*. A small amount of pattern matching lets us guess that the n^{th} element of the sequence is given by:

$$a_n = b + dn.$$

Such a formula is a *closed-form expression*: it explicitly encodes a_n as terms of a function of n. Using such a closed-form expression, one can compute a_n for any arbitrary n without enumerating all intermediate elements in the sequence.

Similarly, consider a geometric progression (numbers of the form $b, rb, r^2b, \ldots$), defined using the recursion:

$$a_0 = b,$$
$$a_n = a_{n-1}r.$$

To compute a closed-form expression for a_n, we can use forward substitution, as defined above. However, let's now try a slightly different approach. Start with a_n, and repeatedly invoke the recursion, i.e.,

$$\begin{aligned} a_n &= a_{n-1}r. \\ &= a_{n-2} \cdot r \cdot r \\ &= a_{n-3} \cdot r \cdot r \cdot r \\ &\dots \\ &= a_0 \, r^{\text{n times}} \cdots r \\ &= br^n. \end{aligned}$$

This method is called *backward substitution*.

Let's go back the savings bank account example from earlier. Suppose we invest d dollars every month; moreover, the (monthly) appreciation ratio is r (i.e., at the end of each month the value gets multiplied by a factor r). Assuming our initial investment is d, we can conclude that the balance in the account at the n^{th} month, a_n, is given by:

$$\begin{aligned} a_0 &= b, \\ a_n &= ra_{n-1} + b. \end{aligned}$$

Now, it would be convenient if we had a closed form expression for a_n. We can use backward substitution as follows:

$$\begin{aligned} a_n &= ra_{n-1} + b \\ &= r(ra_{n-2} + b) + b = r^2 a_{n-3} + rb + b \\ &= r^2(ra_{n-3} + b) + rb = r^3 a_{n-3} + r^2 b + rb + b \\ &= \dots \\ &= r^n a_0 + r^{n-1} b + r^{n-2} b + \dots + rb + b. \end{aligned}$$

Now, we can use the base case and plug in $a_0 = b$. Simplifying, we get:

$$a_n = b(r^n + r^{n-2} + \dots + r + 1).$$

Therefore, a_n is nothing but b multiplied by the sum of all powers of r up to n. Below, we will see how to simplify such sums even further.

Before we conclude this section, here is an **exercise**. Can you model the following sequences using a recursion, and if possible, guess a closed form expression?

$3, 9, 27, 81, 327, \dots$

$1, 2, 6, 24, 120, 720, \dots$

$1, 1, 2, 3, 5, 8, \dots$

(The last sequence has a special name: it is called a *Fibonacci sequence*, and has some fascinating and surprising applications. Look it up!)

Summations

Consider any sequence $(a_0, a_1, a_2, \ldots, a_n)$. The *summation* of this sequence is given by:

$$S_n = a_0 + a_1 + \ldots a_n \quad = \sum_{i=0}^{n} a_i.$$

The lower and upper limits (below and above the Σ respectively) indicate the set of elements that participate in the summation.

Important note #0: any summation S_n itself is a sequence. (Observe that the value of S_n depends on n, which is an integer.)

Important note #1: any summation S_n can be defined using recursion! Here is the setup; the logic should be clear from the (literal) definition of summation.

$$S_0 = a_0$$
$$S_n = S_{n-1} + a_n.$$

Here are a few examples of summations.

$$\sum_{i=0}^{n} i = 1 + 2 + \ldots + n.$$
$$\sum_{i=0}^{4} i^2 = 0 + 1 + 4 + 9 + 16 = 30.$$
$$\sum_{i=0}^{3} (-1)^i = 1 + (-1) + 1 + (-1) = 0.$$
$$\sum_{i=3}^{73} i^3 = 3^3 + 4^3 + 5^3 + 6^3 + \ldots 73^3.$$

Important note #2: the limits can be shifted using a change of variable. For example, in the last summation above, if $i = j + 3$, then we can plug in the counter j instead of i to obtain the (completely equivalent) summation:

$$\sum_{j=0}^{70} (j+3)^3 = 3^3 + 4^3 + 5^3 + 6^3 + \ldots 73^3.$$

Note that the upper and lower limits have now changed; be mindful of this whenever you do a change of variables while doing a summation.

Common summations

Having modeled summations as recursions, we now derive *closed* form expressions for some common summations. The first example is in the context of arithmetic progressions. Consider the special case where the base is zero and the step-size is 1, i.e., the sequence is the natural numbers $(1, 2, 3, 4, 5, \ldots)$. Then, the sum of the arithmetic series is given by:

$$S_n = 1 + 2 + \ldots + n - 1 + n.$$

There is a particularly elegant way to simplify this summation. Trivially, one can observe that this is the same as

$$S_n = n + n - 1 + \ldots + 2 + 1.$$

Now, we will add the two equations term-by-term. We get:

$$2S_n = (n+1) + (n+1) + (n+1) + \ldots (n+1) = n(n+1)$$

or, solving for S_n, we get

$$S_n = \frac{n(n+1)}{2}.$$

For general arithmetic progressions $(a, a+d, a+2d, \ldots,)$, one can show (as an **exercise**) that

$$\begin{aligned} S_n &= a + (a+d) + \ldots + (a+nd) \\ &= \frac{n+1}{2}(2a + nd). \end{aligned}$$

(Aside: The sequence S_n is called the sequence of *triangular* numbers. A possibly apocryphal story is that the legendary mathematician Carl Friedrich Gauss figured out this derivation for S_n

...

when he was in primary school.

So there's that.)

The second example is for geometric progressions. Suppose we have base $b = 1$, i.e., we consider the sequence $(1, r, r^2, \ldots)$. We assume that $r \neq 1$. (If $r = 1$, there is a particularly simple expression for S_n; can you guess it?).

Then,

$$S_n = 1 + r + r^2 + \ldots + r^n.$$

Again, there is a simple derivation for the closed form expression of S_n. Multiply the equation on both sides by r. Then we get:

$$rS_n = r + r^2 + r^2 + \ldots + r^{n+1}.$$

Now, we will subtract the first equation from the second equation. Observe that all the "diagonal" terms cancel each other out, giving us:

$$rS_n - S_n = r^{n+1} - 1$$

or, solving for S_n, we get:

$$S_n = \frac{r^{n+1} - 1}{r - 1}.$$

Chapter 15

Summations, Sizes of Infinite Sets

We introduced summations in the previous lecture. Recall that a summation of a sequence is itself a sequence; summations can be computed using the forward or backward substitution methods; and summations of arithmetic and geometric progressions will of key interest in many scenarios.

A couple of final points about summations. First, here is a list of a summation formulas that can be useful to remember (and we will prove a few of these when we study mathematical induction):

(i) $\sum_{i=1}^{n} 1 = n$

(ii) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

(iii) $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(n+2)}{6}$

(iv) $\sum_{i=1}^{\infty} x^i = \frac{1}{1-x}$ if $|x| < 1$

Second, it is perfectly ok to talk about *double* summations, where the counter loops over two indices. For example, an expression of the form:

$$\sum_{i=1}^{10} \sum_{j=1}^{5} ij$$

can be interpreted as

$$\sum_{i=1}^{10} i(1+2+3+4+5) = (\sum_{i=1}^{10} i)15 = \frac{10 \cdot 11}{2} \cdot 15 = 825.$$

where in the last step, we have used formula (ii) listed above for $n = 10$.

Countable sets

Recall that the cardinality of a set S is the number of elements in the set, denoted by $|S|$. Sometimes, we need to deal with infinitely large sets However, a somewhat subtle fact arising from set theory is that not all infinities are created equal, and some infinite sets are "larger" than other infinite sets. But what does this even mean?

Let us be a bit more precise. We will say that any two sets A and B (whether infinite or not) will have the same cardinality iff there is a one-to-one correspondence (i.e., a bijection) between the elements of A and the elements of B.

A *countable* set is defined as any set S that is either finite, or has the same cardinality as the set of positive integers $\mathbb{Z}^+$. If S is not countable, it is said to be *uncountable*. In other words, if we can establish a one-to-one correspondence between S and $\mathbb{Z}^+$, then S is countable.

Here is a different way to think about it. We defined a sequence as any function from (a subset of) the positive integers to a given base set S. A *countable* set would be any set whose elements could be arranged in the form of some sequence.

Here are some examples.

1. The set $\mathbb{Z}^+$ is countable (trivially).
2. The set $\mathbb{Z}$ is also countable. This is because the set of *all* integers can be naturally indexed as follows:

   $$f(1) = 0, f(2) = 1, f(3) = -1, f(4) = 2, f(5) = -2, f(6) = 3, f(7) = -3, \ldots$$

   More precisely,

   $$\begin{aligned} f(n) &= n/2 \quad \text{if n is even,} \\ &= -(n-1)/2 \quad \text{if n is odd.} \end{aligned}$$

   This is a great example of two sets, $\mathbb{Z}$ and $\mathbb{Z}^+$ where clearly $\mathbb{Z}^+$ is a proper subset of $\mathbb{Z}$, but they are of the same size! Such a statement is absurd when talking about finite sets, but funny things happen when we are dealing with infinities.
3. Other examples: the set of all even integers is countable, and so is the set of all odd integers.
4. A more counter-intuitive example: the set of rational numbers $\mathbb{Q}$ is *also* countable. In other words, the cardinality of the set of rational numbers is the same as that of the positive integers. On the other hand, the set of real numbers $\mathbb{R}$ is *uncountable*. Strange! We will leave the proofs of these statements as challenging **exercises** in an upcoming practice problem set.

Chapter 16

Mathematical induction

We now present a powerful proof technique, known as *mathematical induction*. Indeed, this will be perhaps the single most important concept that you would need to pick up in CPRE 310.

Induction, simply put, is a technique that will be used to prove that a given property is true for all nonnegative integers. We will see that induction provides an elegant mechanism to prove seemingly-difficult statements involving sequences, summations, and the like.

The best way to understand the method of induction is using the following example: suppose (after a particularly good showing by the class in the midterm) a hypothetical CPRE 310 professor wants to distribute candy among all the students. The rule by which he distributes them:

- Line up all the students in a sequence.
- The first student gets a piece of candy.
- If a student gets candy, then the next student in line also gets candy.

Simple enough, and it is clear that eventually everyone in line will get candy. Now, suppose we wish to **prove** a statement of the form:

> *Student #116 will get the candy bar.*

Obviously, the statement is true. But how do we (formally) prove it? The idea is to use the following sequence of arguments:

- It is true that Student 0 gets candy.
- Student 0 gets candy $\implies$ Student 1 gets candy.
- Student 1 gets candy $\implies$ Student 2 gets candy.
- ...
- Student 115 gets candy $\implies$ Student 116 gets candy.

If we observe the above reasoning carefully, we realize this is nothing but *modus ponens* applied a bunch of times. In fact, for any n (up to the size of the class), it is true that Student n will receive that candy. Induction is basically an abstraction of this idea.

A good mental picture to have is imagine a sequence of dominoes stacked up one next to the other. If the first domino falls, then eventually the n^{th} domino also will fall for any $n \geq 1$.

The principle of induction

Let $P(n)$ be a predicate defined on the nonnegative integers. If

- (Base case) $P(0)$ is true, and
- (Inductive step) $P(n) \implies P(n+1) \quad \forall n \geq 0$

then

- $P(m)$ is true for all nonnegative integers.

In terms of predicate logic, we can write the induction principle as the following *rule of inference*:

$$\frac{\begin{array}{c} P(0) \\ \forall k \geq 0,\ P(k) \implies P(k+1) \end{array}}{\therefore\ \forall n,\ P(n)}$$

To be more precise, the above rule of inference is called "ordinary induction." Later we will talk about a different flavor of induction called "strong induction" which is a variant of the above idea.

The validity of the above rule may seem fairly obvious. However, this is essentially the structure of all induction proofs; establish the *base case* ($P(0)$) and establish the *induction step* $P(k) \implies P(k+1)$ for any arbitrary k.

A few examples will show how truly powerful induction is.

Induction: An example

Recall that the summation of the first n nonnegative integers is given by the following closed expression:

$$0 + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}.$$

Previously, we used some Gauss-inspired mathematical trickery to prove this statement. Let us now prove it in a completely different way using induction.

Let $P(n)$ be the predicate that the summation of the first n numbers is $n(n+1)/2$. We need to prove that $P(n)$ is true for all values of n.

To do this, we need to establish two simple arguments:

- (Base case) Show that $P(0)$ is true. To prove this, we observe for $n = 0$, the left hand side of the summation is zero, while the right hand side is $0(0 + 1)/2 = 0$. Therefore, the statement $P(n)$ is true for $n = 0$.
- (Induction step) Now consider $P(k)$ for **some generic** $k \geq 0$. We need to show that if $P(k)$ is true then necessarily $P(k + 1)$ also has to be true. Suppose $P(k)$ is true, i.e.:

  $$0 + 1 + 2 + \ldots + k = \frac{k(k + 1)}{2}$$

  Starting from here, we need to somehow deduce that $P(k + 1)$ is true, i.e.,

  $$0 + 1 + 2 + \ldots + k + k + 1 = \frac{(k + 1)(k + 2)}{2}$$

  In order to do this, we start from the left hand side of the first expression for $0+1+\ldots k$, add $k + 1$ to both sides, and simplify:

  $$\begin{aligned} 0 + 1 + \ldots + k + (k + 1) &= \frac{k(k + 1)}{2} + (k + 1) \\ &= \frac{k(k + 1) + 2(k + 1)}{2} \\ &= \frac{(k + 1)(k + 2)}{2} \end{aligned}$$

  which is precisely the right hand side of the second expression. Therefore, if $P(k)$ is true, then so is $P(k + 1)$.

Combining the above arguments, the principle of induction tells us that $P(n)$ is true for all nonnegative integers n.

Template for induction proofs

A (well-written) proof of induction will have the following essential components.

- Begin by saying "We prove this statement by induction."
- Define an appropriate **predicate** $P(n)$. This is called the *induction hypothesis.* Identifying the correct induction hypothesis is the most important part of any induction proof; a well-modeled induction hypothesis can greatly simplify the problem, while a poorly modeled hypothesis can lead you down a rather deep rabbit hole. Typically, the induction hypothesis pops out of the statement, but some other times you need to carefully model it based on the problem that you are trying to solve.
- Prove that the **base case** $P(0)$ is true. Clearly mark it. End this part by stating "This establishes the base case."
- Show that the **induction step** is true, i.e., use your assumption that $P(n)$ is true to show that $P(n + 1)$ is also true. Usually this involves the most work, but End this part by stating "this completes the induction step."
- Conclude by invoking the induction principle. "By induction, $P(n)$ is true for all n."

More proofs by induction

Let us prove 3 different examples of inductive proofs. They are all pretty similar.

1. Suppose we wish to prove that for all $n \geq 0$,

   $$n < 2^n.$$

   Let $P(n)$ be the predicate that $n < 2^n$. Clearly, $P(0)$ is true since $0 < 2^0$. Suppose we assume that $P(k)$ is true for some k, i.e., $k < 2^k$. Then, adding 1 on both sides, $k + 1 < 2^k + 1$. However, $1 < 2^k$ and therefore

   $$2^k + 1 < 2^k + 2^k = 2^{k+1}.$$

   Therefore, combining the two inequalities, we get $k + 1 < 2^{k+1}$, which means that $P(k + 1)$ is also true. Therefore, by induction, $P(n)$ is true for all n.

2. Suppose we wish to prove that $n^2 - n$ is even for all $n \geq 0$. Let $P(n)$ be the predicate that $n^2 - n$ is zero. Clearly, $P(0)$ is true since $0^2 - 0 = 0$ is even. Suppose that $P(k)$ is true for some k, i.e., $k^2 - k$ is even. We need to show that $P(k)$ is true, i.e., $(k + 1)^2 - (k + 1)$ is even. Simplifying,

   $$(k + 1)^2 - (k + 1) = k^2 + 2k + 1 - k - 1 = k^2 + k.$$

   But $k^2 + k = (k^2 - k) + 2k$. We already know that $k^2 - k$ is even (by the induction hypothesis) and $2k$ is even (by definition of even), and therefore k^2+k is even. Therefore, by induction, $P(n)$ is true for all n.

3. Suppose we wish to prove that the sum of the first n *odd* positive integers is given by n^2.

   $$1 + 3 + 5 + \ldots + 2n - 1 = n^2.$$

   We leave this as a straightforward (but interesting) **exercise**.

Faulty induction proof

Using induction, let us now prove a statement that is obviously false:

> *All horses in the world have the same color.*

Here is the "proof".

Let $P(n)$ be the predicate that the number of distinct colors in any set of $n \geq 1$ horses in the world is exactly 1. Then $P(n)$ is true for all n.

- (Base case) $n = 1$ is trivially true since any set containing 1 horse contains exactly 1 color.
- (Induction hypothesis) Consider an arbitrary set of n horses. If $P(n)$ is true that then they are all of the same color.

- (Induction step) We have to show that an arbitrary set of $n+1$ horses, $\{H_1, H_2, \ldots, H_n, H_{n+1}\}$ have the same color. Consider the first n horses $H_1, \ldots, H_n$. By the induction hypothesis, they are of the same color. Therefore, H_1 is of the same color as $H_2, \ldots, H_n$. Consider the last n horses $H_2, \ldots, H_{n+1}$. Again, by hypothesis they are of the same color. Therefore,, H_{n+1} is the same color as $H_2, \ldots, H_n$.
- Therefore the $\{H_1, H_2, \ldots, H_{n+1}\}$ contains exactly 1 color. Done!

Where is the fallacy? Clearly the statement is false since we have black, brown, and white horses in the world.

The fallacy lies in the following (very subtle) flaw in the induction step. The base case is for $n = 1$. However, the induction step *only* works for $n \geq 2$; otherwise, there is no meaning to the statement "H_1 is of the same color as $H_2, \ldots, H_n$".

In terms of symbolic logic, we have proved that $P(1)$ is true, and we have proved that $P(n) \implies P(n+1)$ for $n \geq 2$. But the crucial link $P(1) \implies P(2)$ is missing, and therefore the entire argument falls apart.

Moral of the story: be careful while doing induction!

Chapter 17

Induction (contd.)

While proving a statement by induction, it is often fruitful to ask yourself the following questions:

- Is the domain of discourse the nonnegative integers (or some subset of the nonnegative integers).
- Does the statement hold for the first few cases?
- If an oracle told us that the statement is true for k, can we use that fact to prove it to be true for $k+1$?

In order to construct a proof by induction, the answer should be yes in each of these cases.

Some more examples

Let us do a couple more induction examples.

First example. Suppose we want to prove that the summation of a geometric series (with ratio $r \neq 1$) is given by:

$$1 + r + \ldots + r^n = \frac{r^{n+1} - 1}{r - 1}.$$

Again, previously we had given a proof by mathematical jugglery. Now we prove it by induction.

Let $P(n)$ be the assertion that the summation has the above form.

(base case) Clearly $P(0)$ is true since the left hand side is equal to 1, while the left hand side is given by $\frac{r^{0+1}}{r-1} = 1$.

(induction) Suppose that $P(k)$ is true for some $k \geq 0$, i.e.,

$$1 + r + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1}.$$

Adding r^{k+1} on both sides, we get:

$$\begin{aligned} 1 + r + \ldots + r^k + r^{k+1} &= \frac{r^{k+1} - 1}{r - 1} + r^{k+1} \\ &= \frac{r^{k+1} - 1 + r^{k+1}(r-1)}{r-1} \\ &= \frac{r^{k+1} - 1 + r^{k+2} - r^{k+1}}{r-1} \\ &= \frac{r^{k+2} - 1}{r-1}. \end{aligned}$$

Therefore, $P(k+1)$ is true, and we are done.

Second example. Suppose we have a robot moving about on a grid. Let its coordinates be denoted as (x, y). In each time step, the robot makes a *diagonal* move, i.e., it can move to $(x+1, y+1)$ or $(x+1, y-1)$ or $(x-1, y+1)$ or $(x-1, y-1)$. Here is a question:

> *Suppose the robot starts at* $(0, 0)$*. Is it possible for the robot to ever reach* $(63, 56)$*?*

Reachability is an important concept in robotic path planning. We prove that $(63, 56)$ is not a reachable location. In fact, we prove the following, more general claim:

> *If* (x_n, y_n) *is the location of the robot after* n *steps. If the robot starts at* $(0, 0)$ *then* $(x_n + y_n)$ *is an even integer for all* n*.*

We prove this by induction. Let $P(n)$ be the assertion that $x_n + y_n$ is even.

(base case) P(0) is true since $x_0 + y_0 = 0 + 0 = 0$, which is an even integer.

(induction) Assume that $P(k)$ is true, i.e., $x_k + y_k$ is even for some $k \geq 0$. We need to show that $P(k+1)$ is true. Due to the rules of transition of the robot, we have exactly 4 cases:

1. $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k + 1)$. In this case, $x_{k+1} + y_{k+1} = x_k + y_k + 2$ which is an even number plus 2. Therefore, $x_{k+1} + y_{k+1}$ is even.
2. $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k - 1)$. In this case, $x_{k+1} + y_{k+1} = x_k + y_k$ which is even.
3. $(x_{k+1}, y_{k+1}) = (x_k - 1, y_k + 1)$. In this case, $x_{k+1} + y_{k+1} = x_k + y_k$ which is even.
4. $(x_{k+1}, y_{k+1}) = (x_k - 1, y_k - 1)$. In this case, $x_{k+1} + y_{k+1} = x_k + y_k + 2$ which is an even number minus 2. Therefore, $x_{k+1} + y_{k+1}$ is even.

Therefore, in any case, $P(k+1)$ is true. Thus the claim is proved by induction.

The principle of strong induction

The method of induction we have seen until now is actually a particular flavor known as *ordinary* induction.

We now discuss a second flavor known as *strong induction*. The method is a slight rearrangement of ordinary induction, and the structure for the proof is exactly the same. The only difference is that instead of assuming that $P(k)$ is true and proving $P(k+1)$ is true, you are free to assume that $P(1), P(2), \ldots, P(k)$ is true, and use all of this information to prove that $P(k+1)$ is true.

The goals of ordinary and strong inductions are exactly the same. However, strong induction is useful when the truth of $P(k+1)$ does not follow directly from the truth of $P(k)$, but requires that $P(l)$ is true for *all* integers $l \leq k$.

Here is a more formal description. Let $P(n)$ be a predicate defined on the nonnegative integers. If:

- (Base case) $P(0)$ is true, and
- (Inductive step) $P(1) \wedge P(2) \wedge \ldots \wedge P(k) \implies P(k+1) \quad \forall k \geq 0$

then

- $P(n)$ is true for all nonnegative integers.

In terms of predicate logic, we can write the induction principle as the following rule of inference:

$$\frac{\begin{array}{c} P(0) \\ \forall k \geq 0,\ P(1) \wedge \ldots \wedge P(k) \implies P(k+1) \end{array}}{\therefore\ \forall m \geq 0,\ P(n)}$$

Examples

Let us do a few examples that use strong induction.

We first prove the *Binary representation theorem*, which is stated as follows:

> Every positive integer n can be written as $n = b_r 2^r + b_{r-1} 2^{r-1} + \ldots + b_1 2 + b_0$.

Simply put, this states that every integer n (bigger than 0) can be written out in binary form $(b_r b_{r-1} \ldots b_1 b_0)$; for example, 14 can be written as (1110) in binary.

We prove this theorem by strong induction. Let $P(n)$ be the assertion that n has a binary representation.

(base cases) The base cases $P(1)$ and $P(2)$ are immediate since 1 is 1 in binary, and 2 is 10 in binary.

(strong induction hypothesis) Suppose that $P(1), P(2), .., P(k-1), P(k)$ are true.

(induction step) We need to prove that $P(k+1)$ is true. We have exactly two cases:

- $k+1$ is an even number. By definition $k+1 = 2p$ for some integer $p \leq k$. By the strong induction hypothesis, $p = b_r 2^r + b_{r-1} 2^{r-1} + \ldots + b_1 2 + b_0$. Therefore $k+1 = 2p = b_r 2^{r+1} + b_{r-1} 2^r + \ldots + b_1 2^2 + b_0 2$. Thus, $k+1 = (b_r b_{r-1} \ldots b_0 0)$ in binary.
- $k+1$ is an odd number. By definition $k+1 = 2p+1$ for some integer $p < k$. By the strong induction hypothesis, $p = b_r 2^r + b_{r-1} 2^{r-1} + \ldots + b_1 2 + b_0$. Therefore $k+1 = 2p+1 = b_r 2^{r+1} + b_{r-1} 2^r + \ldots + b_1 2^2 + b_0 2 + 1$. Thus, $k+1 = (b_r b_{r-1} \ldots b_0 1)$ in binary.

In either case, $k+1$ has a binary representation, and $P(k+1)$ is true.

Here is a second example. Suppose we wish to prove the following:

> Suppose we have two types of postage stamps worth 10 cents and 15 cents respectively. Then, we can exactly make any amount $\geq$ 10 cents that is a multiple of 5 cents.

In other words, any denomination that is a multiple of 5 cents can be represented using some combination of 10-cent and 15-cent postage stamps. Let $P(n)$ be the assertion that $5n$ cents can be realized in this manner. We need to show that $P(n)$ is true for all $n \geq 2$.

(base cases) The base cases $n = 2$, $n = 3$ are immediate since we can represent $10c$ and $15c$ using 1 postage stamp each.

(strong induction hypothesis) Assume that $P(2), P(3), .., P(k-1), P(k)$ is true.

(induction step). To show that $P(k+1)$ is true, we observe that $5(k+1) = 5(k-1) + 10$. But we know $P(k-1)$ is true due to the strong induction hypothesis, i.e., we can represent $5(k-1)$ using some particular combination of stamps. Therefore, we can represent $5(k+1)$ by adding one $10c$ stamp to that combination.

Therefore by strong induction, $P(n)$ is true for all $n \geq 2$.

Chapter 18

Counting

Enumerating the elements of a set is central to several problems in computer engineering (and several other technical disciplines.)

However, counting can be hard. Consider, for example, writing an AI software program that can play a game of chess versus a human. However, even if the program were to think four or five moves "ahead", the set of possible moves that the program must consider can become incredibly complex. Often, it will help if we can establish a collection of simple principles/tools for counting, and this will also help us start reasoning about problems in a quantitative manner.

What does it even mean to *count* the elements of a set S? We briefly touched upon this when we discussed sequences. The idea is to construct a bijective mapping between the elements of S and a subset of the integers (say A), and sets of integers are easy to count. We know due to properties of bijective functions, the cardinality of S and A must be the same. This is the basic principle that we use in all of counting.

However, the above principle is somewhat abstract-sounding. From a practical standpoint, counting problems can be solved using a few simple rules involving addition and multiplication. The hard part is to know *when* to add and *when* to multiply. The best way to understand this is to by giving some examples.

Addition rule and its generalizations

The first rule of counting is called the *addition rule*, or the *sum rule*. Suppose that there are two finite sets (say A and B) that are mutually exclusive, i.e., $A \cap B$ is empty. Then, the number of ways to choose an element of $A \cup B$ is given by $|A| + |B|$.

First example:

> Bart owns five bicycles and three cars. How many different ways can he get to work?

Let B denote the set of bicycles owned by Bart, and C denote the set of cars. The total set of travel options he has is given by $B \cup C$. By the sum rule, the number of ways for Bart to

choose a method to go to school is given by $|B \cup C| = |B| + |C| = 5 + 3 = 8$.

Second example:

> Lisa has decided to shop at exactly one store today, either in the north part of town or the south part of town. If she visits the north part of town, she will shop at either a mall, a furniture store, or a jewelry store. If she visits the south part of town then she will shop at either a clothing store or a shoe store. How many stores could she end up shopping in?

We use an identical argument as above: let S be the set of shops in the south part of town, and N be the set of shops at the north part of town. Then, the set of shopping choices for Lisa is given by $N \cup S$. By the sum rule, $|N \cup S| = |N| + |S| = 3 + 2 = 5$.

Third example:

> The Frying Dutchman (tFD) in Springfield has 25 customers for lunch and 37 for dinner. How many unique people visited tFD?

Here is an attempt to answer this question. Let L be the set of customers for lunch, and D be the set of customers for dinner. Then by the sum rule, $|L \cup D| = |L| + |D| = 25 + 37 = 62 \ldots$

However, the above answer is **incorrect**. We did not specify whether L and D were disjoint or not; it could be that any subset of the 25 lunch customers also returned for dinner, in which case the number of *unique* visitors will vary. Be careful while applying the sum rule.

The sum rule is stated above for 2 sets, but there is a straightforward generalization to n sets. Here is the *generalized addition rule*, stated in full:

> Suppose $A_1, A_2, \ldots, A_n$ are mutually exclusive sets. Then $|A_1 \cup A_2 \ldots A_n| = |A_1| + |A_2| + \ldots |A_n|$.

The proof of this statement is an easy exercise by **induction**.

Product rule and its generalizations

The next rule of counting is called the *product rule*. If A and B are finite sets then recall that $A \times B$ is the Cartesian product of A and B, containing all ordered pairs (a, b) where $a \in A$ and $b \in B$. Then, the number of elements in $A \times B$ is $|A| \times |B|$.

> Bart owns five bicycles and three cars. He plans to drive up to the base of a trailhead and then bike up a mountain. How many different ways can he do this?

As above, let $B = \{b_1, b_2, b_3, b_4, b_5\}$ be the set of bikes, and $C = \{c_1, c_2, c_3\}$ be the set of cars, owned by Bart. Any combination that Bart uses to go to the top of the mountain can be denoted as (b, c) where $b \in B$ and $c \in C$. Therefore, the total number of choices used by Bart is given by the number of possible ordered pairs (b, c) – and by the product rule, this is equal to $|B \times C| = |B| \times |C| = 15$.

As with the sum rule, one can generalize the product rule to n sets. Here is the *generalized product rule* stated formally:

> Suppose $A_1, A_2, \ldots, A_n$ are finite sets. Then $|A_1 \times ...A_n| = |A_1| \times \ldots |A_n|$

Let's do a second example application of the product rule.

> Within the Springfield area code (636), each phone number is 7 digits, with the first digit being any number except 0 or 1. How many distinct phone numbers are possible?

Each phone number can be written as (636) $a_1a_2a_3a_4a_5a_6a_7$, where $a_1 \in A_1 = \{2, 3, \ldots, 9\}$ and $a_i \in A = \{0, 1, \ldots, 9\}$ for $2 \leq i \geq 7$. Therefore, by the product rule, the total number of distinct phone numbers is given by $|A_1| \times |A|^6 = 8 \times 10^6 = 8,000,000$.

Third example.

> Lisa sets a computer password. A valid password can contain between six and eight symbols. The first symbol must be a letter (which can be lowercase or uppercase), and the remaining symbols must be either letters or digits. How many different passwords are possible?

The solution to this problem uses *both* the sum and product rules. In fact, this is how most counting exercises will be structured. First, we observe that there are three mutually exclusive cases: Lisa's password either contains 6 symbols, or 7 symbols, or 8 symbols.

Let F denote the set of valid first symbols. Then, $F = \{a, b, \ldots, z, A, \ldots, Z\}$, and $|F| = 26 + 26 = 52$.

Let S denote the set of valid symbols for subsequent positions. Then, $S = \{a, \ldots, z, A, \ldots, Z, 1, \ldots, 9\}$ and $|S| = 26 + 26 + 10 = 62$.

Let the notation S^n denote $S \times S \times \ldots S$ (n times). If the password contains 6 symbols, the set of possible choices is $F \times S^5$. Therefore, by the product rule $|F \times S^5| = |F| \times |S| \times |S| \times |S| \times |S| \times |S| = 52 \times 62^5$.

If the password contains 7 symbols, by an identical argument as above we get $|F \times S^6| = |F| \times |S|^6 = 52 \times 62^6$.

If the password contains 8 symbols, we get $|F \times S^7| = |F| \times |S|^7 = 52 \times 62^7$.

Therefore, by the sum rule, the *total* number of possible passwords is $52 \cdot 62^5 + 52 \cdot 62^6 + 52 \cdot 62^7 = 1.86 \times 10^{14}$.

One point about the product rule is that the sets A_1 and A_2 are allowed to be dependent on each other. Here is an example: suppose we modify the above problem where passwords are only allowed to contain 6 characters, all characters being the *same* digit, i.e., the only allowable passwords are of the form 000000, 111111, and so on. (Admittedly these are not very strong passwords.)

Here is how we apply the product rule for this problem. Again, let F be the first set of valid first symbols. Then, $F = \{0, 1, \ldots, 9\}$, and $|F| = 10$. However, once we choose the first digit, the set S is a *singleton* set since the second digit has to be equal to the first digit. For example, if the first digit was 7, then $S = \{7\}$. In general, S depends on how we choose the first digit but in all cases, $|S| = 1$.

Therefore, by the product rule, the number of allowable passwords $= 10 \times 1 \times 1 \times 1 \times 1 \times = 10$.

Principle of Inclusion-Exclusion

We have (briefly) seen the Principle of Inclusion-Exclusion (PIE) before; this rule is also called "subtraction" rule, and is an extension of the sum rule in the case of non-disjoint sets.

The PIE states that for any two sets A and B (disjoint or not), the following relation holds:

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

Here is an example. Suppose there are 350 applicants for a job at Compu-Global-Hyper-Mega-Net. You are given the following information:

150 are CPRE majors.

80 are business majors.

30 are double-majors in CPRE and business.

How many applicants to the job *did not* major in either CPRE or business?

This is a a straightforward application of PIE. Let C be the set of CPRE majors and B be the set of business majors applying for the job. Then $|B| = 80, |C| = 150$, and $|B \cap C| = 30$. Therefore, by the principle of exclusion, the number of applicants that are either CPRE or business majors is given by $|B \cup C| = |B| + |C| - |B \cap C| = 150 + 80 - 30 = 200$.

Therefore, the number of applicants that are neither CPRE nor business majors is $\overline{B \cup C} = 350 - 200 = 150$.

Chapter 19

Counting (contd)

Having discussed the addition and product rules and the PIE, we can apply them to pretty much any counting problem in discrete math. The best way to become familiar with these rules is (as always) by getting practice and solving problems.

Permutations

Consider any set S. A *permutation* of S is a sequence that uses contains every element of S exactly once. For example, the set $S = \{a, b, c\}$ has the following permutations:

- (a, b, c)
- (a, c, b)
- (b, a, c)
- (b, c, a)
- (c, a, b)
- (c, b, a)

One can check by hand that the six sequences above represent all the possible permutations of $\{a, b, c\}$. But in fact, we can *prove* that there are 6 permutations of a 3-element set, using a version of the Product rule. Observe that there are exactly 3 choices for the first element in the sequence; once we fix the first element, there are exactly 2 choices for the second element; and once the first two elements are fixed, there is only 1 choice for the third element. Therefore, the total number of permutations is $3 \cdot 2 \cdot 1 = 6$.

Here is a slightly different problem.

> Suppose I have fridge magnets in the shapes of the letters A through Z. How many different 3-letter strings can I make?

Since I have only one magnet per letter, I cannot repeat letters. Therefore, the first character in the string has 26 choices; the second only has 25; and the third has 24. Therefore, the total number of choices is $26 \cdot 25 \cdot 24$.

This argument can be generalized into something that we call the *Arrangement Principle*. This principle is stated as follows:

> The number of ways to form a sequence of r distinct elements that are drawn from a set of n elements is given by:
>
> $$P(n, r) = n \cdot (n-1) \cdot (n-2) \cdots (n-r+1)$$
>
> .

In particular, the number of ways to arrange all n elements of the set into a sequence is given by $P(n, n) = n \cdot (n-1) \cdots 2 \cdot 1 = n!$, where ! denotes the factorial function. (By convention, we denote $1! = 1$ and $0! = 1$.)

Notice that the above expression for $P(n, r)$ can be simplified as:

$$P(n, r) = \frac{n!}{(n-r)!}.$$

Let us do some more examples of the Arrangement Principle.

> From Opening Day to September 1, every MLB team can carry up to 25 players on their roster. How many different ways are there to choose a 9-man batting order?

The answer to this is simply $P(n, r)$, where $n = 25$ and $r = 9$. This equals $25!/(25-9)! = 25 \cdot 24 \cdots 17 \cdot 16 \approx 4.7 \times 10^{11}$.

We should be a bit careful while applying the Arrangement principle; people often confuse it with the multiplication rule. Here is an example that explains the difference:

> A jar contains 10 ping-pong balls of distinct colors. Four balls are drawn in sequence. How many ways are there to do this, if: (i) the balls are replaced before the next one is drawn? (ii) the balls are drawn and not replaced?

Let D_1, D_2, D_3, D_4 be the set of choices for each of the 4 draws. In the first case, there are always 10 balls in the urn, so there are always 10 choices for each draw. By the product rule, the number of ways to draw 4 balls equals $10^4 = 10,000$.

However, n the second case, the balls are *not* replaced, therefore the number of choices decreases by one each time a ball is drawn. Therefore, the number of possible ways $= P(10, 4) = 10 \cdot 9 \cdot 8 \cdot 7 = 5,040$.

Division rule

Suppose we want to count the number of people in a room. We could do it the standard way; however, an equally valid way would be to count the number of *ears* in the room, and divide by two! A third way would be to count the number of *toes* in the room, and dividing by 10. (Assume for simplicity that everyone in the room has two ears, and 10 toes.)

This may sound like a dumb way of counting the number of people in the room, but in fact is an instance of a more general counting principle known as the *division rule*.

Define a *d-to-1* mapping as an onto function $f : A \to B$ which maps exactly d elements in the domain to every element in the co-domain. In the above example, the function that maps ears to people is a 2-to-1 mapping, and the function that maps toes to people is a 10-to-1 mapping.

Then, the division rule states that:

> If $f : A \to B$ is a $d - to - 1$ function, then $|A| = d \cdot |B|$.

Again, this should make intuitive sense. Here is an application of the division rule in action:

> King Arthur wants to seat his Knights at the Round Table. The seats are not numbered. How many different ways of arranging n Knights is possible? Two seatings are considered the same if the sequence of Knights in clockwise order starting from Knight 1 is the same.

One can imagine seating the Knights as follows: generate a random permutation of the n Knights, and seat them in that order. Let A denote the set of all possible such permutations. Therefore, at first glance, it seems that the number of arrangements in this case is precisely the number of permutations of n elements, i.e., $|A| = P(n, n) = n!$.

However, this would be a case of *overcounting*. For example, if $n = 4$, denote the Knights by $\{k_1, k_2, k_3, k_4\}$. Then, (k_1, k_2, k_3, k_4) is the same permutation as (k_2, k_3, k_4, k_1). (Indeed, these are also equivalent to (k_3, k_4, k_1, k_2) and (k_4, k_1, k_2, k_3)).

In particular, every cyclic shift of the above sequence is the same seating. Therefore, we can define an $n - to - 1$ function $f : A \to B$, where A denotes the set of all permutations of the n Knights, and B denotes the set of permissible arrangements around the round table. By the division rule, the number of circular arrangements is given by $|A|/n = n!/n = (n - 1)!$.

[As an **exercise**, convince yourself that this is the case by explicitly enumerating all possible circular arrangements with $n = 4$ Knights.]

Chapter 20

Counting (contd)

Having discussed permutations and the Arrangement Principle, we now move to a different class of counting problems that we call *combinations*. But first ...

Some facts about factorials

Recall that we calculated the number of arrangements of r items out of a set of size n as $P(n, r) = n!/(n - r)!$. The ! function has several interesting properties, that we list below. It will be good to become familiar with these facts (and commit them to memory – they are very important):

- By definition, $n! = n \cdot (n - 1) \cdot (n - 2) \ldots 2 \cdot 1$.
- By convention, $0! = 1$.
- Some common values of $P(n, r)$ are useful to know:

$$P(n, n) = n!,\ P(n, 0) = 1,\ P(n, 1) = n.$$

Combinations and the Selection Principle

Before we formally introduce combinations, here is a slight variation of a problem that we saw earlier:

> From Opening Day to September 1, every MLB team can carry up to 25 players on their roster. How many different ways are there to choose a 9-player team?

Recall that the number of ways to pick a *specific starting order* is $P(25, 9)$. However, the above problem is different from the one in the previous lecture since we don't have to specify a starting *order* – only a starting 9. So how do we estimate this new quantity?

One way to solve it is using the division rule. You can think of picking a 9-player starting order as a 2 step process: (1) pick a *set* of 9 starters; (2) line up the 9 starters in order.

Once you pick a set of 9 starters, there are *precisely* $P(9,9) = 9!$ ways to choose a particular order with these players; this is simply an application of the Arrangement Principle with $n = 9, r = 9$. Therefore, if $k = 9!$, then one can imagine a k-to-1 onto mapping from the set of starting orders to the set of 9 starters (without order specified.)

Therefore, by the Division rule, we get that the number of starting orders equals the number of ways to pick a starting $9 \times k$, where $k = 9!$. In other words, our desired answer $= P(25,9)/9! = 25!/((25-9)! \cdot 9!) = 25!/(16!\,9!)$.

In fact, for picking r items out of a set of size n, we will denote this quantity two special symbols: $C(n,r)$ or $\binom{n}{r}$. The following counting rule formalizes this intuition.

> *Selection Principle*: The number of ways to choose a subset of r elements from a set of n elements is
> $$\binom{n}{r} = \frac{n!}{(n-r)!r!}$$

The quantity $\binom{n}{r}$ is read as "n choose r", and is called a *Binomial coefficient*. (The reason for such a name will become clear later.)

More about combinations

Here are some other facts that you probably should commit to memory:

- $C(n,n) = 1$. In words, there is only one way to pick all n elements from a set of size n.
- $C(n,0) = 1$. In words, there is only one way to pick an empty set from a set of size n.
- $C(n,1) = n$. In words, there are exactly n ways to pick 1 element out of a set of size n.
- $C(n,2) = n(n-1)/2,\ C(n,3) = n(n-1)(n-2)/6$.

People often get tripped up as to whether to use $P(n,r)$ or $C(n,r)$ for counting problems. Here is a useful mnemonic: *In arrangement problems, order matters; in selection problems, it doesn't.*

Another problem. Suppose there are 10 empty buckets and 7 identical balls. How many ways are there to distribute the balls into the buckets such that each bucket gets at most 1 ball?

Since buckets can have either 0 or 1 ball, precisely 7 of the buckets are non-empty once we do the distribution. Therefore, the above number is equal to the number of ways of choosing the 7 (lucky) buckets that get the balls. By the Selection Principle, this is nothing but $\binom{10}{7} = 120$.

Note that we could have solved the above problem slightly differently; we could instead have chosen the 3 *unlucky* buckets that don't get any balls, and given each of the other buckets 1 ball each. Therefore, the number of such possibilities is $\binom{10}{3}$, which *also* equals 120.

This is a more general property of combinations: the number of *selecting* r elements out of n is precisely the same as the number of *rejecting* $n - r$ elements out of n. In other words,

$$\binom{n}{r} = \binom{n}{n-r}.$$

(You could have perhaps also derived this using the definition of $\binom{n}{r}$. But the above argument is clean, completely devoid of messy algebra, and a perfectly legitimate proof technique called "counting-in-two-ways", which we discuss below.)

Some more examples

Let's do a couple of more examples of applying the Selection Principle in problems. A *Manhattan Grid* is a network of roads laid out in a perfect grid. Suppose I stand at $(1, 1)$ located at the Top-Left-Corner, and want to go to the destination grid position (r, c), located at the Bottom-Right-Corner. I would like to get there as soon as possible. A *shortest path* on the Manhattan Grid is simply a sequence of right-moves and down-moves that get me to (r, c) as quickly as possible. Here is the counting problem:

> Assuming a perfect grid with c columns and r rows, How many shortest paths are there from (Top-Left-Corner) to (Bottom-Right-Corner)?

Concretely: suppose $r = 4, c = 3$. Let us model the above problem as follows. Denote each "down" move as D, and each "right" move as R (and "up" as U and "left" as L). Then, the following strings would represent shortest paths:

- DDRDR
- RRDDD
- DDDRR

On the other hand, DDDDRRRULRLRLRL would *not* be a *shortest* path since it comprises 13 moves to the destination, and the above listed paths are shorter. So the question is: how many possible shortest paths exist?

Staring at the above examples (and the candidate shortest paths) a bit, we arrive at the following key insight:

> Every shortest path consists of precisely $r - 1$ "down" moves and $c - 1$ "right" moves.

This insight is enough to give us the answer! This is because every string that represents the shortest paths has to have $r - 1$ "D"s and $c - 1$ "R"s; however, the order of these "R"s and "D"s don't matter.

Therefore, each such path is specified by picking the $c - 1$ locations of the "R"s in a string of total length $r - 1 + c - 1 = r + c - 2$. The final number we need is

$$\binom{r + c - 2}{r - 1}.$$

One more example.

> The Indians and the Cubs are in the World Series, which ends when one team wins 4 games. How many possible win/loss scenarios exist in which the Cubs end up winning?

This problem is a bit more tricky, and involves a combination of counting rules.

The first insight is that there are 4 cases: the series ends either with the Cubs winning either 4-0, or 4-1, or 4-3, or 4-3. Moreover, these cases are mutually exclusive! (This is usually a suggestion that the Sum Rule should be used.)

The second insight is that the last (n^{th}) game in the series is always a Cubs win, and the previous $n-1$ games either have 0 Indians wins, or 1, or 2, or 3 (depending on how many games the series goes.)

Therefore, we solve the problem by applying the Selection Rule to each of the sub-cases, and then the Sum Rule overall.

- If the series goes 4 games, there are $\binom{3}{0} = 1$ such possibility.
- If the series goes 5 games, there is 1 Indians win in the first 4 games, and there are $\binom{4}{1} = 4$ such possibilities.
- If the series goes 6 games, there are 2 Indians wins in the first 5 games, and there are $\binom{5}{2} = 10$ such possibilities.
- If the series goes 6 games, there is 3 Indians win in the first 6 games, and there are $\binom{6}{3} = 20$ such possibilities.

Therefore, overall there are $1 + 4 + 10 + 20 = 35$ win/loss scenarios.

(As an **exercise**, try counting the number of win-loss scenarios overall, regardless of whether the Indians or the Cubs win. You should get an answer of 70.)

Combinatorial proofs

Above, we proved the following identity:

- $\binom{n}{r} = \binom{n}{n-r}$

However, the proof was obtained by counting a certain specific set of possibilities in two different ways, and equating the results. This is an instance of a *combinatorial proof*, and is often used in proving identities that involve factorials and binomial coefficients.

A typical combinatorial proof of a given identity proceeds as follows. Construct some set A whose size equals the left hand side; show that A is equal to some other set B whose size equals the right hand side. Done!

Sounds fairly arcane, so let's do a few examples. Suppose we wish to prove that:

- $\binom{n+1}{r} = \binom{n}{r} + \binom{n}{r-1}$

One can attempt to prove this directly using algebra, or using induction on n (although either might get a bit messy).

Here is a far cleaner proof, using counting-in-two-ways. Imagine that we have a class of $n+1$ CPRE students with Jill being one of them. Suppose the class decides to form a committee of r students. By the Selection Principle, there are $\binom{n+1}{r}$ ways to choose this committee. That is the left hand side of the above identity.

Moreover, observe that every such committee has either Jill as a member, or does not. These are mutually exclusive cases.

If Jill isn't a member, then there are r empty slots in the committee and n candidates (the remaining students) to choose from. We can do this in $\binom{n}{r}$ ways. If Jill is a member, then we need to fill the $r-1$ remaining slots from the n students. We can do this in $\binom{n}{r-1}$ ways.

Therefore, using the Sum Rule, the total number of ways to pick the committee is $\binom{n}{r} + \binom{n}{r-1}$. This is the right hand side of the identity we wish to prove. Done!

As **exercises**, try proving the following identities using combinatorial proofs. (Follow the same process as above.)

- $\binom{m+n}{r} = \sum_{i=0}^{r} \binom{m}{i}\binom{n}{r-i}$.
- $\sum_{i=0}^{n} \binom{n}{i} = 2^n$.

Chapter 21

Counting (contd)

Counting in two ways

Combinatorial proofs usually involve counting the elements of some set in two different ways, and equating the results. This notion of "counting-by-two-ways" is a general technique used in counting problems.

We saw a few examples of combinatorial proofs in previous lecture. Here is one more. Let us try to prove the following identity:

$2^n = \sum_{i=0}^{n} \binom{n}{i}$.

As with all counting proofs, we need to construct a counting problem, calculate the answer in two different ways, and equate the results.

The left hand side of the identity should be familiar by now: 2^n is the number of subsets of a set of size n. This should give us some idea of where to start.

So here is the proof: consider the problem of counting the number of subsets of a given set X containing n elements. We know that this number equals 2^n.

However, we can also count the number of subsets in a different way. Any given subset can have size 0, or 1, or 2, ... or n. Moreover,

- The number of subsets of size 0 is $\binom{n}{0}$.
- The number of subsets of size 1 is $\binom{n}{1}$.
- The number of subsets of size 2 is $\binom{n}{2}$.
- ...
- The number of subsets of size n is $\binom{n}{n}$.

Therefore, by the sum rule (since the above cases are mutually exclusive), we get that the total number is given by $\sum_{i=0}^{n} \binom{n}{i}$. Equating this to our first answer (2^n), we complete the proof.

A different, *algebraic* proof of the above identity can be derived as follows. From elementary algebra, we know the following facts about polynomials:

- $(1+x)^2 = 1 + 2x + x^2$
- $(1+x)^3 = 1 + 3x + 3x^2 + x^3$
- $(1+x)^4 = 1 + 4x + 6x^2 + 4x^3 + x^4$
- ...

In fact, these are all special cases of a more general result called the *Binomial theorem*:

$$(1+x)^n = \sum_{i=0} \binom{n}{i} x^i.$$

which itself can be derived using induction or other means. Plugging in $x = 1$ in the above formula, we get the desired identity.

Here is a (somewhat) more challenging **exercise**: can you come up with a combinatorial proof of the following equation?

$\binom{n+1}{r+1} = \sum_{j=r}^{n} \binom{j}{r}$.

(A hint to get you started: imagine selecting a subset of size $r+1$ given a set of n numbers, and focus on the *biggest* number that you select.)

Counting with repetitions

Counting rules are by-and-large simple applications of logic. But the most common pitfalls occur when we have a counting problem where some of the objects under consideration are repeated, i.e., are *indistinguishable* from the others. In such cases we need to be a bit careful when applying the counting rules.

Consider the following problem:

> Suppose you are planning a 20-mile walk, which should include 5 northward miles, 5 eastward miles, 5 southward miles, and 5 westward miles. How many different walks are possible?

The first step in solving such problems is to translate into a setting more suitable for enumeration. Strings of characters are often used for this purposes. For example, for the above problem let us represent each walk as a string of length-20 consisting of 5 occurrences of "N", "E", "S", and "W" each. We need to count the number of such possible strings.

We can now solve this counting problem as follows. Imagine 20 empty slots for the characters and filling in these slots with 5 "N"s, 5 "E"s, 5 "W"s, and 5 "S". First, we can place the locations of the "N"s in

$$\binom{20}{5} = \frac{20!}{15! \cdot 5!}$$

ways. Once we do that, there are 15 empty slots remaining. We can now place the "E"s in

$$\binom{15}{5} = \frac{15!}{10! \cdot 5!}$$

ways. There are 10 empty slots remaining. We can choose the "W"s in

$$\binom{10}{5} = \frac{10!}{5! \cdot 5!}$$

ways and the remaining necessarily have to be "S"s. Therefore, the total number is given by:

$$\begin{aligned}\binom{20}{5}\binom{15}{5}\binom{10}{5} &= \frac{20!}{15! \cdot 5!}\frac{15!}{10! \cdot 5!}\frac{10!}{5! \cdot 5!} \\ &= \frac{20!}{5! \cdot 5! \cdot 5! \cdot 5!}.\end{aligned}$$

More generally, if we want to permute n objects of which k_1 objects are of type 1, k_2 objects are of type 2, and so on – the

$$\frac{n!}{k_1! \cdot k_2! \ldots k_n!}$$

A proof of this result is identical to that in the above example with the 20-mile walk. We leave it as an **exercise**.

Here is a different example.

> How many different ways are there to select 10 donuts if there are 4 varieties to choose from?

One can solve it using a variant of the above method, but the catch here is that before you knew exactly how many occurrences of each type were present in your selection, while here it can be anywhere between 0 and 10.

A generalization of this example would be:

> We have r varieties of donuts and need to choose a set of n donuts in total. In how many ways can we do this?

Let's focus on the 10-donut-4-variety case first. A fairly ingenious way of solving this problem is as follows. Suppose we choose 3 chocolate donuts, 4 Boston kremes, 2 lemon donuts, and 1 plain donut. We can imagine writing down our choice of donuts as a *binary string* as follows:

0001000010010

In the above string, 0's denote donuts, and 1's denote dividers between different types of donuts. Note that we have exactly 10 0's (10 types) and 3 1's (4-1=3 dividers between different types.) On the other hand, if all 10 of our selected donuts were Boston kremes, then our choice would be written as

1000000000011

Again, a length-13 string with 10 0's and 3 1's! Convince yourself there is a *one-to-one* bijection between the set of possible choices of 10 donuts and 4 varieties, and the set of binary strings of length-13 with 3 1's.

Therefore, by the Selection Principle, the answer is immediate; the number of donut selections is given by:

$$\binom{13}{3}.$$

For general problems for selecting n objects from r varieties, we will have a binary string with n 0's and $r-1$ 1's. The answer, using analogous reasoning as above, is given by:

$$\binom{n+r-1}{r-1} = \binom{n+r-1}{n}.$$

Pigeon Hole Principle

We now discuss a somewhat surprising rule in counting, one that is fairly obvious. Here is an example problem:

> A drawer in a dark room contains red socks, green socks, and blue socks. How many socks must you withdraw to be sure that you have a matching pair?

Thinking about this a bit, we realize that the answer is 4. If we pick 3 socks or less, there is a chance that there is no matching pair. However, if we draw 4 socks, then we are sure to get at least one matching pair.

This kind of reasoning is called the *Pigeon Hole Principle*, which is the following aphorism:

> If there are more pigeons than holes, then at least two pigeons must be in the same hole.

What's our colorful sock collection got to do with pigeons and holes? Imagine the socks that we pick out represent "pigeons", and the colors represent "holes". Then, if we pick 4 socks (pigeons) then by the Pigeon Hole Principle at least two of them are of the same hole, i.e., the same color.

Here is a more generic application of the Pigeon Hole principle. We claim that:

> There are 112 students enrolled in CPRE 310. Some group of at least 12 students will get the same letter grade in the end.

How do we arrive at this? Well, here the "holes" are the letter grades: (A,A-,B+,B,B-,C+,C,C-,D,F), and the "pigeons" are CPRE students. So if there are 10 holes it has to be that some hole gets more than 11 pigeons; if not, the number of pigeons is at most 110 (which is not true in this case).

This idea is nothing but the *Generalized* Pigeon Hole Principle, stated in set-theoretic language as follows:

> If $f : A \to B$ and $|A| > k|B|$, then f maps some $k+1$ elements of A to the same element in B.

One caveat of the Pigeon Hole Principle is that it is non-constructive; the Principle does not give you any indication of *which* pigeons get mapped to which holes – only that there exist some pigeons that get mapped to the same hole. This can have important practical implications. For example, consider the problem:

> Suppose we generate a list of 40 10-digit numbers at random. Prove that there exist some two distinct subsets of these numbers that add up to the same value.

This is called the *Subset-Sum* problem, and has several practical applications (including in privacy and cryptography). It is hard to explicitly *find* two different subsets that add up to the same value, but not hard to argue that two such subsets *exist*.

The solution is surprisingly simple. Let A be the set of all subsets (i.e., the power set) of these 40 numbers. The number of subsets (pigeons) of the list of numbers is given by:

$$|A| = 2^{40} > 10^{12}.$$

On the other hand, let B be the set of sums that you can generate from adding up subsets. We don't know the precise set, but we can safely say that $|B| < 40 \times 10^{10}$ since each 10-digit number is less than 10^{10} and we can add up at most 40 of them. Therefore, $|B| < 4 \times 10^{11}$.

Since $|A| > |B|$, by the Pigeon Hole Principle, some two pigeons (subsets) get mapped to the same hole (subset sums). Done!

Chapter 22

Counting (fin)

Counting with recurrences

The *Tower of Hanoi* problem is a famous problem in combinatorics. Typically one introduces the problem using a picture of 3 pegs with a stack of disks of increasing sizes – something like this:

Figure 22.1: Tower of Hanoi (Source: Wikipedia)

Suppose we start with all the disks on Peg 1 (the leftmost one), arranged in increasing disk-radius from top to bottom.

The goal is to move the disks one by one until they are all on Peg 2, while satisfying the Smaller-on-Top Rule. This Rule states that at no point in time can you place a bigger disk on top of a smaller disk. Now suppose we we want to answer the question:

How many moves are needed to shift a stack of n disks over to a different peg?

In order to count the number of minimum moves, let us recall a very clever *recursive* algorithm to solve this problem. The idea is that if there was some method/routine (say ROUTINE) to solve the problem for n disks, then one can solve the problem for $n+1$ disks by the following algorithm:

- applying ROUTINE to move the top n disks from Peg 1 to Peg 3.
- moving the largest disk from Peg 1 to Peg 2.
- applying ROUTINE to move the n disks from Peg 3 to Peg 2.

If Routine doesn't violate the Smaller-on-Top Rule, then the above recursive algorithm doesn't either. Moreover, the base case of the recursion is for $n = 1$, where one can (trivially) move the lone disk over from one peg to another. Therefore, the above algorithm is correct.

Given the recursive structure of the algorithm, we can analogously *count* the number of moves as follows: let H_n be the number of moves to solve the Tower of Hanoi problem with n disks. Then, the number of moves to solve it for $n + 1$ disks (adding up the number of moves for the 3 steps above) is given by:

$$H_{n+1} = H_n + 1 + H_n = 2H_n + 1.$$

The base case is $H_1 = 1$. Plugging in a few numbers, we obtain $H_2 = 3, H_3 = 7, H_4 = 15, \ldots$. As an easy **exercise**, prove (using induction and/or simplifying geometric progressions) that a closed form expression for H_n is given by:

$$H_n = 2^n - 1.$$

Discrete probability

One last topic on counting. We won't have time to go very deep into probability theory (and this is not how you would learn it in a statistics class), but the very basics of probability can be directly understood using counting.

Assume that we perform some type of procedure or experiment. The set of all possible outcomes of the experiment is called the *sample space*. An *event* is a specific subset of the sample space.

For instance, if we toss a coin three times, the set of possible outcomes (i.e.the sample space) is given by:

$$S = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}.$$

The event "All Heads" would correspond to the (singleton) subset:

$$A = \{HHH\}.$$

The event "At least two heads in a row" would correspond to the subset:

$$A = \{HHH, HHT, THH\}.$$

If all possible outcomes of an experiment have the same likelihood of occurrence, then the probability of an event $A \subseteq S$ is given by:

$$P(A) = \frac{|A|}{|S|}.$$

Applying the above formula, the probability of getting at least two heads in a row in the above experiment is 3/8, whereas the probability of getting "All heads" is 1/8.

In general, computing the probability of an event will involve (carefully) enumerating the number of elements in the subset that defines the event, and dividing by all possible outcomes.

Some properties of probabilities:

- By definition, since A is a subset of S, we always have $0 \leq P(A) \leq 1$. A probability value that is negative, or bigger than 1, is meaningless.
- For every event A, the probability of $\bar{A}$ (its complement) is given by:

$$P(\bar{A}) = 1 - P(A).$$

- By the addition rule of counting, if A_1 and A_2 are mutually exclusive events, then we know that $|A_1| + |A_2| = |A_1 \cup A_2|$. Therefore,

$$P(A_1) + P(A_2) = P(A_1 \cup A_2).$$

- By the Principle of Inclusion-Exclusion, for general A_1 and A_2, we know that $|A_1| + |A_2| = |A_1 \cup A_2| - |A_1 \cap A_2|$. Therefore,

$$P(A_1) + P(A_2) = P(A_1 \cup A_2) - P(A_1 \cap A_2).$$

Example. *Craps* is a game played in casinos, where players roll a pair of ordinary (6-sided) dice. If the total of the numbers on the rolled dice on the first roll is 2, 3, or 12, the person "craps out", i.e., they lose the bet. If the total is 7 or 11, they win the bet. In other cases, the game continues (and other rules kick in).

Given this information, we can compute some probability of *winning* on the first roll as follows:

- There are 6 possibilities on the first die and 6 on the second die. Therefore, the total sample space has cardinality $6 \times 6 = 36$.
- Let E_7 be the event that we get a seven. There are precisely 6 ways in which we can do this $(6+1, 5+2, 4+3, 3+4, 2+5, 1+6)$. Therefore, $P(E_7) = 6/36 = 1/6$.
- Let E_{11} be the event that we get an eleven. There are precisely 2 ways in which we can do this ($6+5$ and $5+6$). Therefore, $P(E_{11}) = 2/36 = 1/18$.
- Since E_7 and E_{11} are mutually exclusive, the probability of winning on the first roll is given by $P(Win) = P(E_7) + P(E_{11}) = 1/6 + 1/18 = 4/18 = 2/9$.

Analogously, as an **exercise**, try computing the probability of losing on the first roll. (You should get 1/9.)

Chapter 23

Relations

Binary Relations

Having defined sets, functions, and so on, we now introduce *relations*. Relations capture *pairwise interactions* between objects, and are a powerful generalization of the concept of a function.

First, a formal definition. Let A and B be sets. Then, a *binary relation* R, defined from A to B, is a subset of $A \times B$, i.e.,

$$R \subseteq A \times B$$

In other words, R is a set of ordered pairs (a, b) where $a \in A$ and $b \in B$

The set A is called the *domain* of the relation, The set B is called the *co-domain* of the relation.

To denote that some object $a \in A$ is related to some other object $b \in B$, we will use the notation:

$$aRb, \quad \text{or,} \quad (a, b) \in R$$

The above definition is a bit abstract, but some examples might help to clarify. Suppose that A denotes the set of all men in the world, and B is the set of all women. Suppose R denotes the relation "is the husband of", then some subset of ordered pairs in $A \times B$ belong to R.

Another example. Suppose that P denotes the set of all webpages and L is a relation between two different webpages a and b if a links to b. Then, L can be written as:

$$L = \{(a, b) \in P \times P \mid a \text{ has a link to } b\}$$

Relations are even easier to understand if we are talking about numerical objects. Let

$$A = \{0, 1, 2\}, \quad B = \{1, 2, 3\}$$

be two sets of numbers. Let R denote the "is less than or equal to" relation, i.e,

$$xRy \iff x \leq y.$$

Then, we can assert that

$$0R1, 0R2, 0R3, 1R2, 1R3, 2R3$$

i.e., R is the set of of ordered pairs:

$$\{(0,1),(0,2),(0,3),(1,1),(1,2),(1,3),(2,2),(2,3)\}$$

On the other hand, if R denotes the "equal to" relation, then we can assert that:

$$1R1,\ 2R2$$

Not surprisingly, such a relation is called the *equality* relation and is commonly encountered in several applications.

Functions as relations

If we stare at the definition of a relation a bit carefully, then we realize that *every function can be interpreted as a relation.* Let

$$f : A \to B$$

be an arbitrary function from A to B. We can always define a relation $F \subseteq A \times B$ which consists of ordered pairs (a, b) such that $b = f(a)$ for a given a.

Example. Define $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) = n^2$. Then, the corresponding relation $F \subset \mathbb{N} \times \mathbb{N}$ consists of all pairs of natural numbers $\{(1,1),(2,4),(3,9),(4,16)\ldots,(n,n^2),\ldots\}$.

However, there is an important distinction between relations and functions: relations can include objects of the form (a, b) and (a, c). This is an instance of a one-to-many mapping, where a given object a in A can be related to more than one object in B. So *not every relation is a function.*

A good example of this effect is the "Circle" relation. Consider the graph of the algebraic equation corresponding to the unit circle:

$$x^2 + y^2 = 1.$$

We have previously seen that this is *not* a function in the strict sense, since the value $x = 0$ can be mapped to both $y = 1$ and $y = -1$. However, it *is* a relation, constructed as follows. Define $A = B = \mathbb{R}$ and define a relation C as follows: $(x, y) \in C$ iff the Cartesian coordinates (x, y) lie on the unit circle. Therefore, C is a relation but not a function.

Mathematically representing relations

We introduce two approaches for mathematically representing binary relations.

The first representation of relations is via *matrices.* Consider any relation R defined from A to B, where A and B are finite sets. Now construct a matrix M with $|A|$ rows and $|B|$ columns, with:

- rows corresponding to elements of A, and

- columns corresponding to elements of B.

Now, if a is related to b (i.e., aRb), then the (i,j)th entry of M ($M_{i,j}$) with row index i corresponding to a, and the column index j corresponding to b, is set as 1; if a and b are not related, then $M_{i,j}$ is set to be 0.

Let's start with a simple example. Suppose $A = B = \{1, 2, 3\}$ and we are interested in the "equals" relation R. Construct a 3×3 matrix with rows (and columns) corresponding to 1, 2, and 3 respectively. Then, the matrix representation of R is given by:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Equality relations are often represented as *diagonal* matrices, like the one shown above.

Second example. Consider the set of airports {DSM, ORD, IAH, CLE} and let R denote the relation "exists a direct flight", i.e., aRb iff there exists a direct flight from A to B. We have (a priori) information that there are direct flights from ORD to everywhere else; DSM to ORD (and back); DSM to IAH (and back); IAH to ORD and DSM (and back); CLE to ORD (and back).

Suppose we label the rows and columns as (DSM,ORD,IAH,CLE) in order. As an **exercise**, convince yourself that the relation R can be written as:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

A second representation of binary relations is via *graphs*. We will talk about graphs in the coming lectures, but here is a preview: a graph G consists of a set of nodes V, along with a set of edges E, possibly with arrows (directions) on them. Graphs are a very popular data structure in all kinds of CPRE applications.

Graphs provide a particularly intuitive way to represent relations: given two sets A and B and a binary relation R, we can represent every ordered pair $(a, b) \in R$ by drawing a directed edge (arrow) from a to b. We already drew such "arrow diagrams" to represent functions; relations are represented basically the same way. An important distinction is that relations can potentially have several "outgoing" arrows, while functions can have only 1 outgoing arrow for each element in A.

(Both relations and functions can have elements in B with several "incoming" arrows, however. This is what differentiates onto functions, one-to-one functions, etc.)

Properties of relations

Binary relations are important concepts, and some of them satisfy some interesting properties. There are three main properties that we will need to understand. Let us state them first (assume here that $A = B$ in all the examples below):

1. Reflexivity. A relation R is *reflexive* if aRa.
2. Symmetry. A relation R is *symmetric* if aRb iff bRa.
3. Transitivity. A relation R is *transitive* if for any triple $a, b, c \in A$, aRb and bRc then necessarily aRc.

Some examples for each of these properties. If $A = B = \mathbb{N}$, then the equality relation is reflexive since any number a is equal to itself. Similarly, the "$\leq$" relation is reflexive. On the other hand, the "exists a direct flight" relation between airports is *not reflexive* (since having a direct flight from DSM to itself is meaningless.) As an **exercise**, convince yourself that " $<$ " is not a symmetric relation.

Now let A denote the set of all users on Facebook. The "friend" relation is a *symmetric* relation (if Person i is a friend of j, then j is automatically a friend of i). On the other hand, the "follow" relation is not symmetric; I follow LeBron James, but that does not mean that LeBron James follows me (I think).

Finally: the $=, \leq$ and $<$ relations are all transitive with respect to the set of natural numbers (or real numbers). The logic being: if $a = b$ and $b = c$, then necessarily $a = c$. Likewise for $<$ and $\leq$. On the other hand, let A be the set of sports teams in a league, and R denote the "has defeated" relationship. If Team a has defeated Team b, and Team b has defeated Team c, that does not automatically imply that Team a has defeated Team c. The "has defeated" relation is *not transitive*.

Last example: consider a "ping" relation in a computer network, i.e., Computer i is related to Computer j if i can ping j. Now we know that:

- every computer can ping itself.
- if Computer 1 can ping Computer 2, then Computer 2 can ping Computer 1
- if 1 can ping 2 and 2 can ping 3, then 1 can ping 3.

Therefore, the "ping" relation satisfies all three relations. Any relation that is reflexive, symmetric, transitive is called an *equivalence* relation, which we will study in more detail.

Chapter 24

Relations (contd)

Properties of relations

Recall that we discussed 3 main properties of relations (assuming that the base sets A and B over which the relation is defined are the same):

1. Reflexivity: A relation R is *reflexive* if $\forall a,\ aRa$
2. Symmetry: A relation R is *symmetric* if $\forall a, b,\ \ aRb \iff bRa$.
3. Transitivity: A relation R is transitive if $\forall a, b, c, aRb \wedge bRc \implies aRc$.

Here are some examples of different relations, and some of their properties. As an **exercise**, convince yourself that the stated properties hold:

- $A = B = \mathbb{R}$, $R =$ "$<$". This relation is not reflexive; no symmetric; and transitive.
- $A = B =$ set of positive integers, $R = \{(a, b) | a + b = 4\}$. This relation is not reflexive; symmetric; not transitive.
- $A = B =$ set of all people who ever lived, R: "has the same blood group as". This relation is reflexive; symmetric; and transitive.
- $A = B =$ set of professional tennis players, R: "has a ranking greater than or equal to". This relation is reflexive; not symmetric; and transitive.
- $A = B =$ set of subsets of some universal set U; R: "is a subset of". This relation is reflexive; not symmetric; and transitive.

Interpreting the properties via graphs

The above properties may be a bit tricky, but luckily there are some systematic ways to check whether a given relation satisfies a given property. The most intuitive way to check for a property is to look at the graph representation of the relation.

Recall that a graph has nodes, representing objects, and (directed) edges, representing relationships between objects. A *path* is a sequence of contiguous edges along this graph. A

cycle is a path that starts and ends at the same node.

Given these definitions, we can identify relational properties as follows:

- A reflexive relation has the distinct property that *every node has a self-loop*, i.e., each node a has a cycle of length 1 from a to itself.
- A symmetric relation has the property that *every edge must be two-sided*, i.e., if there exists a "forward" edge from a to b, then necessarily there has to be a "reverse" edge from b to a as well.
- A transitive relation has the property that if there is an edge from a to b, and an edge from b to c, then necessarily there is an edge from a to c.

Other properties of relations

Having defined the 3 main properties, let us also quickly define some other properties that are related to these three. It is easiest to describe them in terms of the graph representation defined above:

- A relation is called *irreflexive* if there are **no** self loops in its graph representation. For example, the "$<$" relation between numbers is irreflexive; so is the "exists a direct flight to" relation between airports.
- A relation is called *symmetric* if the following is true for any pair a and b: if aRb, then $bnotRa$. Moreover, $anotRa$. In terms of graph theoretic terms: no cycles of size 1 or 2 are allowed in the graph representation of R. The "$<$" relation is also asymmetric.
- A relation is called *antisymmetric* if no cycles of size 2 are allowed, but self-loops (cycles of length 1) are OK. The "$\leq$" relation between numbers is antisymmetric; so is the "divides" relation between positive integers. Observe that if R is antisymmetric, if aRb and bRa then $a = b$; this is proved by observing that if $a \neq b$, then R has a cycle of size 2, which contradicts the definition of antisymmetry.

Combining relations

There are three different ways to imagine relations: as a set of ordered pairs, as a (directed) graph, and as a matrix. These are equivalent representations, but each is useful in its own context. It is good to get used to thinking about relations in these 3 different ways.

Since relations are sets, you can always *combine* relations using set operations. So if R_1 and R_2 are two relations defined over sets A and B, then the following are perfectly legitimate definitions:

- $R_1 \cup R_2$
- $R_1 \cap R_2$
- $R_1 - R_2$
- $R_1 \Delta R_2$
- etc.

We can analogously combine more than 2 relations by considering multiple unions and/or intersections.

This is particularly useful in the context of search applications using relational databases. For instance, let's say we are modeling a *social network* with users representing different nodes. Let R_1 be the relation "has known for 10 years", and R_2 be the relation "goes to the same school as". If we are interested in all pairs of users who have known each other for 10 years **and** go to the same school, then we need to consider the relation $R_1 \cup R_2$. However, if we only want pairs of users who go to the same school but have *not* known each other for 10 years, then we need to look at $R_2 - R_1$. So on and so forth.

Similarly, we can *compose* different relations as follows. Let R be a relation from A to B, and S be a relation from B to C. Then $R \circ S$ is defined as a relation from A to C, and consists of all ordered pairs (a, c) such that aRb and bSc.

For example: consider A as a set of people, and let R be the "parent of" relation. Then, $R \circ R$ is the composition of R with itself, and represents the "grandparent" relation (since if a is the parent of b and b is the parent of c, then a is the "grandparent" of c).

Last example. Consider the "exists direct flight" relation between airports (say, R). Then, the composite relation $R \circ R$ represents the relation "exists a 2-hop trip" between airports.

Closure

One last point about relations and properties. Sometimes, a relation may not satisfy a given property, since certain edges/pairs might be missing. The *closure* of a relation with respect to a property can be obtained as follows: if some edges/elements of R are missing, then add them to satisfy that particular property.

The three main types of closure operations are as follows:

- Reflexive closure: If a relation is not reflexive, add all self loops to obtain its reflexive closure. For example, the "$\leq$" relation is the reflexive closure of $<$. In terms of set notation: if Δ denotes the identity relation over any given set (i.e., it consists of self-loops and nothing else), then the reflexive closure of any given relation R is expressed as $R_{refclose} = R \cup \Delta$.
- Symmetric closure: If a relation is not symmetric, make all edges double-sided. If R^{-1} denotes the inverse relation (i.e., the relation with all edges reversed) then the symmetric closure $R_{symclose} = R \cup R^{-1}$. For example, if R denotes "is a parent of", then the symmetric closure of R denotes "is a parent of or is a child of".
- Transitive closure: If a relation is not transitive, then add the "third" edge to every path of length 2 in the directed graph representation of R. You might learn more about closures of different relations in a later Algorithms course.

Chapter 25

Relations (contd)

We have discussed some key properties of relations. We will now define a few more properties; but keep in mind the three key ones: *reflexivity*, *symmetry*, and *transitivity*.

Equivalence relations

How do we mathematically state whether two objects "behave" the same vis-a-vis some given characteristic? This is captured via the notion of *equivalence*.

Equivalence between objects in a given set is an important concept that is naturally captured via relations. Formally, any relation that is

- reflexive
- symmetric
- transitive

is called an *equivalence relation*.

If R is an equivalence relation and aRb, then a is said to be equivalent to b.

Some examples:

1. Define the relation $R = \{(a, b) | a - b \text{ is even}\}$ over the integers. Then, R is reflexive (since for any integer a, we always have $a - a = 0$, which is even). R is symmetric (since $a - b$ is even iff $b - a$ is even.). Moreover, R is transitive (since $a - b$ is even *and* $b - c$ is even implies that $a - c$ is even.) Therefore, R is an equivalence relation.
2. Define the *Congruence* relation over the integers as follows: for a fixed integer m, a is related to b iff $a \mod m = b \mod m$. As an **exercise**, convince yourself that congruence is an equivalence relation.
3. A couple of non-numeric examples. Let A be the set of all cars, and R be the relation defined over A as follows. Let a and b denote cars; aRb iff a and b have the same make+model+year. Then, R is an equivalence relation (prove it as an **exercise**).

4. Let A be the set of all lines on a plane. Let R be the relation defined over A as follows: line a is related to line b iff a and b are parallel. As an **exercise**, prove that R is an equivalence relation.

Every equivalence relation induces one or more *equivalence classes*. An equivalence class is defined as follows: let R be an equivalence relation on A. Then, the set of all elements related to $a \in A$ is called the equivalence class of a, denoted by $[a]$.

$$[a] = \{b | b \in A,\ (b, a) \in R\}$$

Let's go back to the above list of examples for equivalence classes. In the first example, we see that the following elements are all related to each other:

$$\{\ldots, -4, -2, 0, 0, 2, \ldots\}$$

Similarly, the following elements are also related:

$$\{\ldots, -5, -3, -1, 1, 3, 5, \ldots\}$$

In other words, we see there are precisely *two* equivalence classes: the set of all integers related to 0, and the set of all elements related to 1. (These are precisely the even and the odd integers).

Similarly, in the example of cars, the set of all *2003 Ford F150s* forms an equivalence classes; so does the set of all *2014 Honda Accords*. (We see that in this particular example, there is a *very* large number of equivalence classes.)

As an **exercise**, identify the equivalence classes in the other two examples. Given any equivalence relation, you should be able to identify the equivalence classes.

Partitions

A partition is simply a grouping of elements of a given set into disjoint subsets. Formally: given any set A, a partition of A is a collection of subsets, $A_1, A_2, \ldots, A_n$ such that two conditions are satisfied:

- The union of the subsets gives us the whole set: $\bigcup_{i=1}^{n} A_i = A$
- The subsets are disjoint: $A_i \cap A_j = \emptyset$ for all pairs of subsets A_i, A_j for $A_i \neq A_j$.

There is a very tight connection between equivalence classes and *partitions* of a given set. Specifically, we have the following theorem:

> Let R be an equivalence relation defined on A. Then R forms a partition on the sets of A. The subsets in the partition are precisely the equivalence classes induced by R.

Here is a short proof. Given an equivalence relation R over the elements A, we need to somehow produce a collection of subsets which satisfy the above two conditions.

Consider any $a \in A$, and consider the equivalence class $[a]$. Since aRa (due to reflexivity), we know that $a \in [a]$. Therefore, the union of all subsets $[a]$ is *also* the union of all singleton sets $\{a\}$, which is nothing but the whole base set A. Therefore, the first condition is satisfied.

Now consider two elements a and b, and consider their equivalence classes $[a]$ and $[b]$. We need to prove that if $[a] \neq [b]$ then $[a]$ and $[b]$ are disjoint. We will prove this via contraposition. Suppose $[a]$ and $[b]$ are not disjoint. Therefore, there exists some element c that belongs to both $[a]$ and $[b]$, i.e., aRc **and** bRc. By the transitivity property, we have that aRb. Therefore, a and b belong to the same equivalence class, and by definition $[a] = [b]$. By contraposition $[a] \neq [b] \implies [a] \cap [b] = \emptyset$.

Quick example: consider the congruence relation mod 4. This forms a *partition* of $\mathbb{Z}$ into 4 subsets:

- $\{\ldots, -8, -4, 0, 4, 8, \ldots\}$
- $\{\ldots, -7, -3, 1, 5, 9, \ldots\}$
- $\{\ldots, -6, -2, 2, 6, 10, \ldots\}$
- $\{\ldots, -5, -1, 3, 7, \ldots\}$

Partial order relation

Another important class of relations are called *order* relations. Just as how equivalence relations group objects that are "similar" to one another, order relations group objects according to some notion of *hierarchy*.

There are two types of order relations – partial orders and total orders.

Any relation that is

- reflexive
- antisymmetric
- transitive

is called a *partial order*. A set with a partial order relation defined on it is called a *partially ordered set*, or *poset* for short.

Two examples of partial order relations:

1. Consider R as the "$\leq$" relation between natural numbers. It is reflexive (since $a \leq a$ for any number a) and transitive (since $a \leq b$ and $b \leq c$ implies that $a \leq c$). Moreover, it is *antisymmetric*: no loops of length 2 are allowed since $a \leq b$ for distinct a and b implies that $b \not\leq a$. Therefore, R is a partial order.
2. Let A be the set of all subsets (i.e., the power set) of some universe U. Let R be the $\subseteq$ relation, i.e., aRb if $a \subseteq b$ (this is strange notation since we usually use caps to denote subsets, but here the *elements* of A are themselves subsets.) As an **exercise**, prove that the $\subseteq$ relation is (i) reflexive, (ii) antisymmetric, and (iii) transitive. Therefore, R is a partial order.

Example 2 above illustrates an important point:

> *In a partial order, not every pair of elements need to be related.*

In the "subset of" relation, there could well be a pair of sets such that neither is a subset of the other. For example, if $U = \{1, 2, 3\}$, $a = \{1, 2\}$ and $b = \{2, 3\}$, then neither aRb nor bRa.

Note that this goes against the conventional idea of "ordering" some set of objects. If we want to order the set of pro tennis players by assigning them a ranking, then implicitly *every* pair of players can be related this way (i.e., either player X has a higher ranking than player Y, or vice versa.) On the other hand, partial orders are more permissive.

Let R be a partial order relation. If either aRb or bRa, then a and b are said to be "comparable". Else, a and b are *incomparable*.

In contrast, in Example 1 above (i.e., the "$\leq$" relation), all pairs of integers are comparable, i.e., for any pair of elements a, b then necessarily we have either aRb, or bRa, or both. Such a relation is called a *total order*.

Chapter 26

Order relations

Partial and total orders

Recall our definition of a *partial order relation*: any relation R that is reflexive, **anti**-symmetric and transitive is called a partial order relation (and the set on which it is defined is called a partially ordered set, or a poset). Partial order relations capture hierarchies induced among objects of a given set.

By the way, partial order relations are usually denoted by the symbol "$\succeq$" (instead of the generic symbol "R"). If you see a relation being expressed as $a \succeq b$, then we can interpret this as a relational edge between elements a and b, with a being dominated by b in the sense of the given relation.

As discussed in previous lecture, the word *partial* in "partial order relation" denotes the fact that not all elements need to be comparable. For example, if we define the set:

$$A = \{1, 2, 3, 9, 18\}$$

and define R as the "divides" relation (i.e., aRb iff $a|b$). Then we have the following edges in the graph representation of R:

- all elements having self-loops
- an edge from 1 to every other elements
- an edge from 2 to 18
- an edge from 3 to 9, and 3 to 18
- an edge from 9 to 18.

In the above graph, there is *no* edge between 2 and 3, or between 2 and 9. Therefore, 2 and 3 are incomparable; so are 2 and 9.

If **all** pairs of elements in a set are comparable with respect to a given relation, then the relation is called a *total order*, or *linear order*. The "$\leq$" relation between real numbers is a total order (easy to see, since for any generic pair of numbers a and b, either $a = b$, or one is bigger than the other.)

Lexicographic order

Order relations between elements can be naturally extended to order relations between *n-tuples* of elements. This can be done in a couple of different ways: one popular method is called the *lexicographic* order, also called the "dictionary order".

As a motivating example, consider the words *computer*, *man*, *manual*, and *market*. In terms of their appearance in the Oxford Dictionary,

- *manual* precedes *market*
- *man* precedes *manual*
- *computer* precedes *man*

But why is this the case? Here is a mathematical argument. Suppose we define a relation $\succeq$ between *characters*, where $\alpha \succeq \beta$ iff the letter α precedes β in the English alphabet. Then, $c \succeq m$ and therefore *computer* precedes the other three words. Between the rest, the first two characters are the same (ma) but $n \succeq r$, and therefore *man* and *manual* precede *market*.

The above example is for the $\succeq$ relation defined on English letters, but the same can be extended to arbitrary partial order relations. Suppose we are given an order relation $\succeq$ over some set of objects A. Then, we can define the *lexicographic order relation*, $\succeq^n$, over $A \times A \times \ldots \times A$ (i.e., n-tuples composed of elements of A) as follows. Given two n-tuples $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ then

$$(a_1, a_2, \ldots, a_n) \succeq^n (b_1, b_2, \ldots, b_n)$$

if

- $a_1 < b_1$, or
- $a_1 = b_1, \ \wedge \ a_2 < b_2$, or
- $a_1 = b_1, \ \wedge \ a_2 = b_2, \ \wedge \ a_3 < b_3, \ \ldots$

As an **exercise**, try defining the lexicographic ordering of all the points on the two-dimensional plane would look like.

Hasse Diagram

We have discussed the way to represent relations via nodes and edges in a directed graph. However, there is an even more concise way to represent order relations, called a *Hasse* diagram.

The high level idea is as follows: if we know *a priori* that a relation R is an order relation, we immediately know that every node in the graph representation of R has a self-loop due to the reflexivity property of order relations. Moreover, if the edges (a, b) and (b, c) exist then the edge (a, c) must also exist due to the transitivity property of order relations.

Therefore, a (significant) simplification to the graph would be to remove all self-loops, and all transitive edges. Moreover, the convention is to all "smaller" elements

If we do this, we obtain a significantly simplified representation of R called the Hasse diagram. To summarize, Here is the general algorithm to obtain this diagram. Starting from the directed graph representation of R:

- Rearrange the nodes and edges such that all arrows are pointed upwards
- remove all self-loops, i.e., remove all edges of the form (a, a).
- remove all transitive edges, i.e., all edges (x, y) if there is an element $z \in A$ such that $x \succeq z$ and $z \succeq y$.
- remove all arrows from the edges to convert into an undirected graph.

Quick example. As above, let $A = \{1, 2, 3, 9, 18\}$ and define R as the "divides" relation. Then, the directed graph representation would be as in Figure 1.

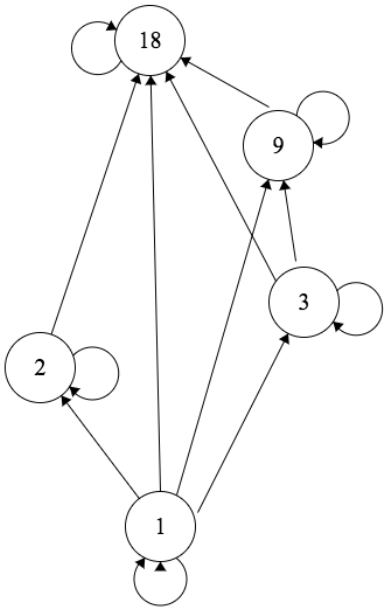

Figure 26.1: Graph representation

Applying the above algorithm to this graph, we remove all self-loops; transitive edges; and arrows, to obtain the Hasse diagram of R. See Figure 2.

Hasse diagrams provide a particularly simple way to check whether an order is a One way to check whether a partial order is a total order or not; write out its Hasse diagram and check whether it's a linear chain. As an **exercise**, try drawing the Hasse diagram of the $\leq$ relation applied to the set $\{1, 2, 3, 4, 5\}$.

Lastly, some more definitions with respect to order relations. A *minimal element* is an element that is not greater than any other element under partial order relation; similarly a *maximal* element is not lesser than any other element. (These need not be unique; there

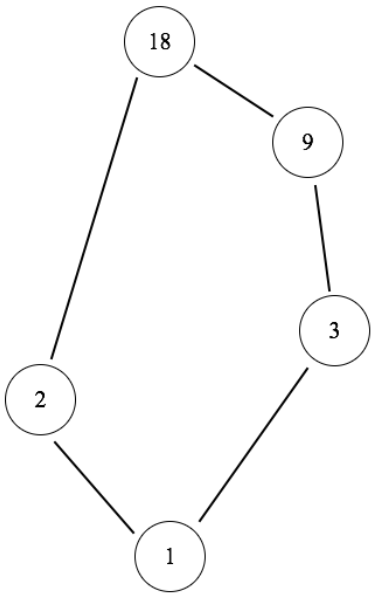

Figure 26.2: Hasse diagram

can be more than 1 minimal or maximal element. If they are unique, then the maximal and minimal elements are respectively called the *greatest* and *least* elements.)

One can easily check that a minimal element is the bottom-most element(s) of a Hasse diagram, while a *maximal element* is the top-most element(s). In the Hasse diagram in this example, the minimal (and least) element is 1, while the maximal (and greatest) element is 18.

Chapter 27

Introduction to Graph Theory

We are transitioning into the last topic of the course (and one of the more important ones) – and that is graph theory. But before we discuss graphs – one last topic with regards to order relations.

Topological sort

The idea of *sorting* a set of jobs/objects is an important component in several CPRE applications, including instruction scheduling in CPUs, compiler design, and concurrency control. In each of these cases, a central controller has a list of *jobs*, and needs to to make decisions about how to schedule different jobs in a consistent manner. The idea is that sorting it *correctly* is important: if Job #1 requires (as input) the output of Job #2, then necessarily #2 must be scheduled such that it is completed before #1 is launched; else disaster ensues.

The catch is that usually in systems, a precise schedule is not given a priori — only a rough specification. It is up to the designer to figure out the rest! That's where Topological Sort (top-sort) for short comes into play.

Top-sort is a way to design a consistent schedule (i.e., a **total order** in which the jobs are performed) given a partial schedule (i.e., a **partial order**).

Formally, top-sort is defined as the following problem:

> given a set of objects and a partial order relation R, produce a total order (i.e., sort the elements) that is compatible with R.

The algorithm for top-sort is surprisingly simple:

1. Draw the Hasse diagram of the partial order relation and set $i = 0$.
2. Pick any minimal element from the Hasse diagram, copy in position i in the total order, remove (pop) from Hasse diagram, increment i.
3. Repeat Step 2 until no more elements remain

Here is a toy (but practical) application. The typical CPRE course schedule has the following prerequisites for different courses:

- ComS 227 for ComS 228
- ComS 228 for CPRE 288
- ComS 228 for CPRE 310
- CPRE 288 for CPRE 310
- CPRE 288 for CPRE 381
- CPRE 310 for CPRE 308
- CPRE 310 for ComS 311

Assuming that a student takes courses one at a time, the goal is to design a consistent (linear) order of the courses such that all prerequisites are satisfied at the time of taking any particular course.

The way to solve this is as follows: (1) model the prereqs as a partial order relation, and draw the directed graph; (2) convert it into the Hasse diagram representation; (3) apply the above algorithm.

We won't do the entire working of the above algorithm (that is left to you as an **exercise**), but one of the outputs of applying top-sort on the above is given as follows (abbreviating course names by numbers):

227, 228, 288, 381, 310, 308, 311

A perfectly valid (but distinct) output would be:

227, 228, 288, 310, 308, 311, 381

This is because the minimal element in a Hasse diagram need not be unique, so different outputs for top-sort may emerge; all that matters is that the order in any given output list respects the constraints given by the input specifications. See if you can produce other potential top-sort outputs for the above

Graphs

We now provide a brief introduction to *graph theory*. Graphs, of course, are one of the most important objects of study in discrete mathematics.

Graphs can be used to study problems ranging from computer networks, to social networks, to the Web, to transportation networks, to biological networks etc etc etc. Suffices to say that applications of graph-based algorithms impact all of engineering.

A graph G is a set of *nodes* V, together with a set of *edges* between nodes E. Concisely, we write this as $G = (V, E)$. Edges can be either equipped with a direction (arrow) or be without any directionality. Depending on this, the graph G is called *directed* or *undirected*.

We have already seen directed graphs in order relations. These can have self-loops, or even double-edges between vertices. *Simple* graphs are special classes of undirected graphs where no self-loops or multiple edges between nodes are allowed.

The *degree* of a node in a simple graph is the number of edges in the graph.

In a directed graph, there are two notions of degree for each node: *in-degrees* (number of incoming edges) and the *out-degree* (number of outgoing edges).

A *path* in an undirected graph is a sequence of edges that connect a sequence of vertices that are distinct from one another.

Let us illustrate these ideas in a concrete example. Consider a simple social network with 9 persons, some of which are friends with each other. We represent the people as nodes, and friendship between people as edges. An example such network is given by Figure 1.

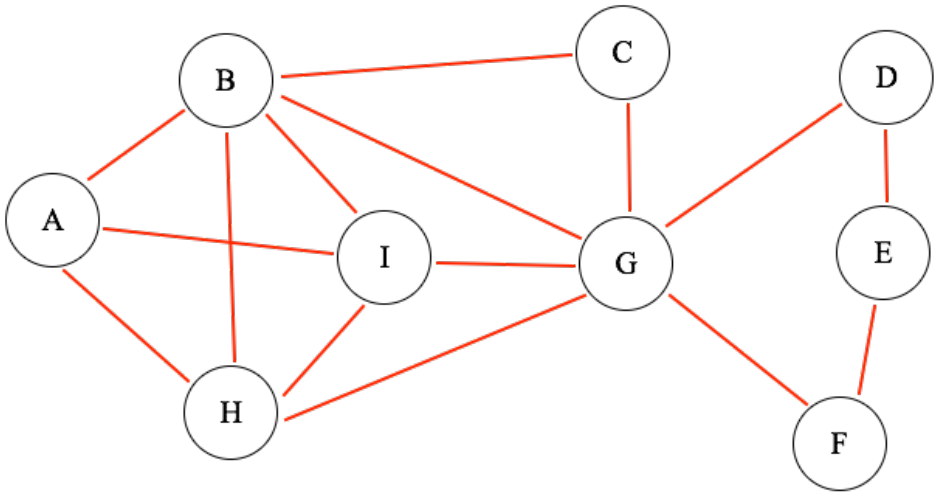

Figure 27.1: Social network

Given this network, let's try to answer some natural questions using the tools of graph theory. Here is the first question:

> A *clique* is a group of friends who all mutually know each other. What is the largest clique in the graph?

We quickly see that there are several triangles (i.e., cliques of size 3) and one group of 4 - $\{A, B, H, I\}$ - which is a clique of size 4. In fact, this is the largest clique in the graph. But how do we *prove* this fact?

We prove that there is no clique of size 5, via contradiction. Suppose, to the contrary, that there exists a clique of size 5. That means that there is a group of 5 people, each of which knows the four other people in the group. Therefore, the degree of each node corresponding to these 5 people is *at least 4*. However, if one enumerates all the degrees in the graph, we get:

- One node of degree 6 (G)
- One node of degree 5 (B)
- Two nodes of degree 4 (H,I)

- One node of degree 3 (A)
- Four nodes of degree 2 (C,D,E,F)

Therefore, it is not true that there are 5 nodes of degree 4. By contradiction, there is is no clique of size 5.

Next question:

> Who is the most "centrally connected" person in the social network?

This is a more qualitative question; however, a natural notion of "central connectivity", again, is the degree of each node. By the above enumeration, G has the highest degree, and therefore, is the most "centrally connected" person.

Last question:

> Suppose the people in the network continue to interact in parties and social events, i.e., they get acquainted with friends of friends, etc. Which two currently unacquainted people are (a) most likely to become acquainted? (b) least likely to become acquainted?

This is a more challenging question. But again, one can solve it using graph theory tools. The trick is to realize that "likeliness of X and Y becoming acquainted" is related to "are there many short paths between X and Y".

Again, by inspection, we see that there are 3 distinct paths of length 2 from A to G (and that this is the highest, i.e., no other pair of people who are currently unacquainted have a higher number of paths of size 2 between them.) Therefore A and G are most likely to be acquainted. Similarly, A and E are least likely to be acquainted since there is only one path of size 5.

One last point: this example involves a fairly simple graph where all the calculations can be done by hand. But imagine posing the same questions, but on a Facebook-scale graph involving (potentially) billions of nodes and edges. How do we reason (mathematically) about such graphs?

This is a vast area of study – but one of immense practical importance that everyone should be aware of. Interesting anecdote: in the late nineties, two graduate students in Stanford (re)discovered a new, ultra-efficient algorithm for estimating "important" nodes of a massive graph (such as the World Wide Web) purely from degree information of the graph. They (Larry Page and Sergei Brin) called the algorithm PageRank, and this algorithm worked very well; so much so that the two founded a small company in the early 00s called *Google*. The rest, as they say, is history.

Chapter 28

Graph Theory

Let us continue our discussion on (undirected) graphs, and develop some basic theory that will let us solve some interesting problems. But first, some history.

Paths, etc.

Let us introduce some terminology. In all of the discussions below, the key idea to keep in mind is that while discussing graphs, it is not at all important *how* the nodes and edges are physically represented, but only *which* nodes are connected to each other. Here, we use the set notation $\{v_1, v_2\}$ to denote an undirected edge between nodes v_1 and v_2 since the order of the nodes do not matter.

Given any graph $G = (V, E)$, we can *traverse* this graph by hopping from node to node along the given edges. There are different types of graph traversals:

- A *walk* is any sequence of hops along a given graph. If the start point is u, and the end point is w, then any walk can be represented as:

  $$ue_1v_1e_2v_2 \dots e_nw$$

  where $e_1, e_2, \dots, e_n$ denote the different edges traversed. In a generic walk, both edges and nodes can be visited one or more times.
- A *path* is any walk that does not contain repeated edges. (However, nodes can be repeated.)
- A *simple path* is any path that does not contain repeated vertices.

Types of graphs

Depending on their connectivity structures, graphs can be classified according to various types. Here are some common graph classes:

- A *line graph* with nodes/vertices $v_1, v_2, \ldots, v_n$ consist of edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, i.e., the nodes are connected sequentially. Line graphs with n nodes have $n-1$ edges.
- A *cycle* with $n \geq 3$ nodes has all the edges as a line graph, with the additional edge $\{v_1, v_n\}$. A cycle is like a "closed loop": one can start at any node and traverse the rest of the nodes sequentially to return to the starting point.
- A *star graph* consists of a "central node" v_1 that is connected to the other "outer" nodes via edges $\{v_1, v_2\}, \{v_1, v_3\}, \ldots, \{v_1, v_n\}$. Star graphs with n nodes have $n-1$ edges.
- A *wheel* with n nodes is formed by starting with a star graph (having $n-1$ edges), and forming a cycle through the $n-1$ "outer" nodes by adding $n-1$ additional edges.
- A *complete* graph with n nodes, commonly denoted by K_n, is formed by connecting each of the n nodes with every other node. Complete graphs with n nodes have $\binom{n}{2}$ vertices; try proving this via a simple counting **exercise**.
- A *tree* is a connected graph containing no cycles. (Here, "connected" means that the graph consists of a single component as opposed to multiple islands, and that there is a well-defined path from every node to every other node.) All trees with n nodes have exactly $n-1$ edges; this is left as another, slightly more involved, **exercise**.

Degree theorems

Recall that the *degree* of any given node v in a graph, denoted by $\deg(v)$, is the number of edges incident on that node. The degree is a simple quantity to compute, but conveys a considerable amount of structural information in the graph. Degrees of nodes in graphs also satisfy a number of interesting properties, which we prove as Theorems below.

First, we prove the *Degree Theorem for undirected graphs*. It states that:

> In any undirected graph,
> $$\sum_{v \in V} \deg(v) = 2|E|.$$

This can be proved using counting in two ways. First, let us focus on the left hand side. For any fixed v, $\deg(v)$ denotes the number of edges connected to v. Therefore, as we enumerate all nodes and add up their degrees, we are listing all the edges present in the graph. However, notice that each edge is counted exactly twice, since every edge is incident to exactly two nodes. Therefore, the sum of the degrees of the vertices is twice the number of edges. This ends the proof.

Second, we prove the *Handshaking Lemma*. It states that:

> In every party, the number of people who have shaken hands an odd number of times is even.

This can be proved using the Degree Theorem. Model people as a set of nodes V, and handshakes as edges E. Then, we get a graph $G = (V, E)$. The degree of each node v is the number of handshakes that person v makes with other people. We partition the people in V into two groups, V_o and V_e, depending on whether people have shaken hands an *odd* or an

even number of times. Splitting the left hand side of the equation in the Degree Theorem into two parts, we get:

$$\sum_{v \in V_o} \deg(v) + \sum_{v \in V_e} \deg(v) = 2|E|.$$

Rewriting, we get:

$$\sum_{v \in V_o} \deg(v) = 2|E| - \sum_{v \in V_e} \deg(v).$$

The right hand side of the equation is even, since $2|E|$ is even and $\sum_{v \in V_e} \deg(v)$ is a sum of even degrees (by definition of V_e), which is also even. Therefore the left hand side of the equation is even. However, the left hand side is the sum of a certain set of *odd* numbers, and can be even only if the number of elements in that set is even. Therefore, the cardinality of V_o is even (or in words, the number of people who have an odd number of handshakes is even.) Done!

In directed graphs (where edges have arrows), there is a slightly more subtle notion of degree: depending on how many arrows are pointing inwards or outwards from a given vertex, we can define an *in-degree* $\deg^+(v)$ as well as an *out-degree* $\deg^-(v)$. The *Degree Theorem for directed graphs* states that:

> In a directed graph,
>
> $$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E|.$$

This is again a consequence of the fact that each edge has precisely one initial vertex and a terminal vertex, and therefore counting the edges will involve adding up all of the in-degrees, or equivalently, all of the out-degrees.

The Seven Bridges of Konigsberg

The field of graph theory (arguably) was kickstarted by Euler when he solved the *Seven Bridges of Konigsberg problem*. Konigsberg (now a city in Russia called Kaliningrad) looks like this today in Google Maps:

but back in the 1700s, there were a few more bridges crossing the rivers. Then, the city looked like this:

There were 7 bridges across the rivers connecting different landmasses, and a friend of Euler posed to him the following question:

> Is it possible to walk around the city such that each bridge is crossed once, and exactly once?

You can try tracing different walks across the bridges yourself. However, you'll find that a walk that does not repeat edges seems difficult to find. In fact, such a walk is impossible, and Euler proved this using graph theory.

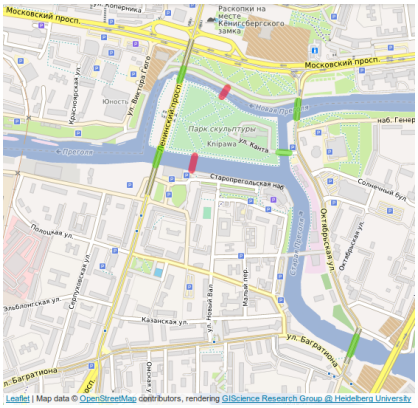

Figure 28.1: Konigsberg today (Google Maps)

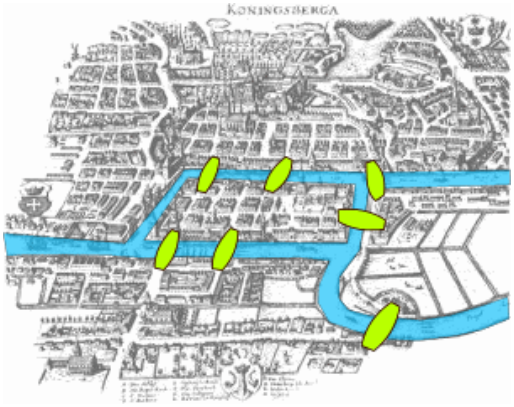

Figure 28.2: Bridges of Konigsberg (Source: Wikipedia)

Euler's main idea (like most great ideas) was deceptively simple:

> Draw the map as a graph.

It does not matter where the bridges were located; all that matters is that we represent landmasses as nodes, and bridges between landmasses as edges connecting nodes. Rewriting the map thus gives us the following graph:

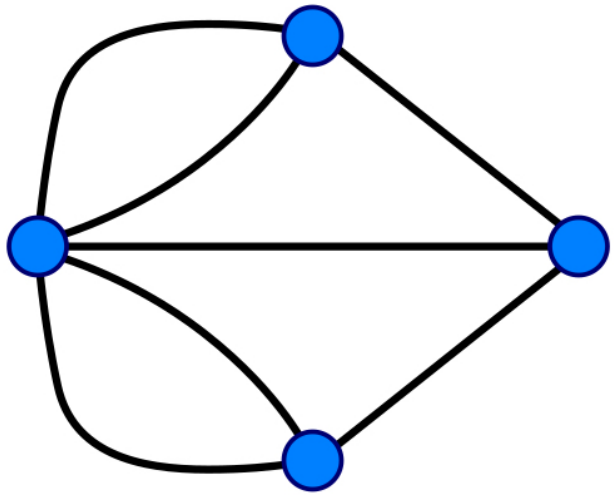

Figure 28.3: Graph of Konigsberg

except the initial and final landmasses, every landmass must be touching an *even* number of bridges.

In the language of graph theory, it is necessary that every node in the path (except first and last) has to have *even degree*.

However, the graph in question has all four nodes of *odd* degree. therefore, no matter what the start and end points were, such a path cannot exist.

Such paths are called *Euler* paths. A necessary condition for an Euler path to exist in a graph is that the graph should be connected, and that the number of nodes with odd degree must be zero or two. In fact, this is also a *sufficient* condition, and this was a neat thing proved by Euler himself.