Preface

Discrete mathematics deals with objects that come in *discrete* bundles, e.g., 1 or 2 babies. In contrast, continuous mathematics deals with objects that vary *continuously*, e.g., 3.42 inches from a wall. Think of digital watches versus analog watches (ones where the second hand loops around continuously without stopping).

Why study discrete mathematics in computer science? It does not directly help us write programs. At the same time, it is the mathematics underlying almost all of computer science. Here are a few examples:

- Designing high-speed networks and message routing paths.
- Finding good algorithms for sorting.
- Performing web searches.
- Analysing algorithms for correctness and efficiency.
- Formalizing security requirements.
- Designing cryptographic protocols.

Discrete mathematics uses a range of techniques, some of which is seldom found in its continuous counterpart. This course will roughly cover the following topics and specific applications in computer science.

1. Sets, functions and relations
2. Proof techniques and induction
3. Number theory
    a) The math behind the RSA Crypto system
4. Counting and combinatorics
5. Probability
    a) Spam detection
    b) Formal security
6. Logic
    a) Proofs of program correctness
7. Graph theory
    a) Message Routing
    b) Social networks
8. Finite automata and regular languages
    a) Compilers

In the end, we will learn to write precise mathematical statements that captures what we want in each application, and learn to prove things about these statements. For example, how will we formalize the infamous zero-knowledge property? How do we state, in mathematical terms, that a banking protocol allows a user to prove that she knows her password, without ever revealing the password itself?

Chapter 1

Sets, Functions and Relations

"A happy person is not a person in a certain set of circumstances, but rather a person with a certain set of attitudes."
– Hugh Downs

1.1 Sets

A set is one of the most fundamental object in mathematics.

Definition 1.1 (Set, informal)**.** A set is an *unordered* collections of objects.

Our definition is informal because we do not define what a "collection" is; a deeper study of sets is out of the scope of this course.

Example 1.2. The following notations all refer to the same set:

$$\{1,2\}, \{2,1\}, \{1,2,1,2\}, \{x \mid x \text{ is an integer}, 1 \leq x \leq 2\}$$

The last example read as "the set of all x such that x is an integer between 1 and 2 (inclusive)".

We will encounter the following sets and notations throughout the course:

- $\emptyset = \{\ \}$, the empty set.
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, the non-negative integers
- $\mathbb{N}^+ = \{1, 2, 3, \dots\}$, the positive integers
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2 \dots\}$, the integers
- $\mathbb{Q} = \{q \mid q = a/b,\ a, b \in \mathbb{Z},\ b \neq 0\}$, the rational numbers
- $\mathbb{Q}^+ = \{q \mid q \in \mathbb{Q},\ q > 0\}$, the positive rationals
- $\mathbb{R}$, the real numbers
- $\mathbb{R}^+$, the positive reals

Given a collection of objects (a set), we may want to know how large is the collection:

Definition 1.3 (Set cardinality)**.** The cardinality of a set A is the number of (distinct) objects in A, written as $|A|$. When $|A| \in \mathbb{N}$ (a finite integer), A is a finite set; otherwise A is an infinite set. We discuss the cardinality of infinite sets later.

Example 1.4. $|\{1, 2, 3\}| = |\{1, 2, \{1, 2\}\}| = 3$.

Given two collections of objects (two sets), we may want to know if they are equal, or if one collection contains the other. These notions are formalized as set equality and subsets:

Definition 1.5 (Set equality)**.** Two sets S and T are equal, written as $S = T$, if S and T contains exactly the same elements, i.e., for every x, $x \in S \leftrightarrow x \in T$.

Definition 1.6 (Subsets)**.** A set S is a subset of set T, written as $S \subseteq T$, if every element in S is also in T, i.e., for every x, $x \in S \rightarrow x \in T$. Set S is a *strict* subset of T, written as $S \subset T$ if $S \subseteq T$, and there exist some element $x \in T$ such that $x \notin S$.

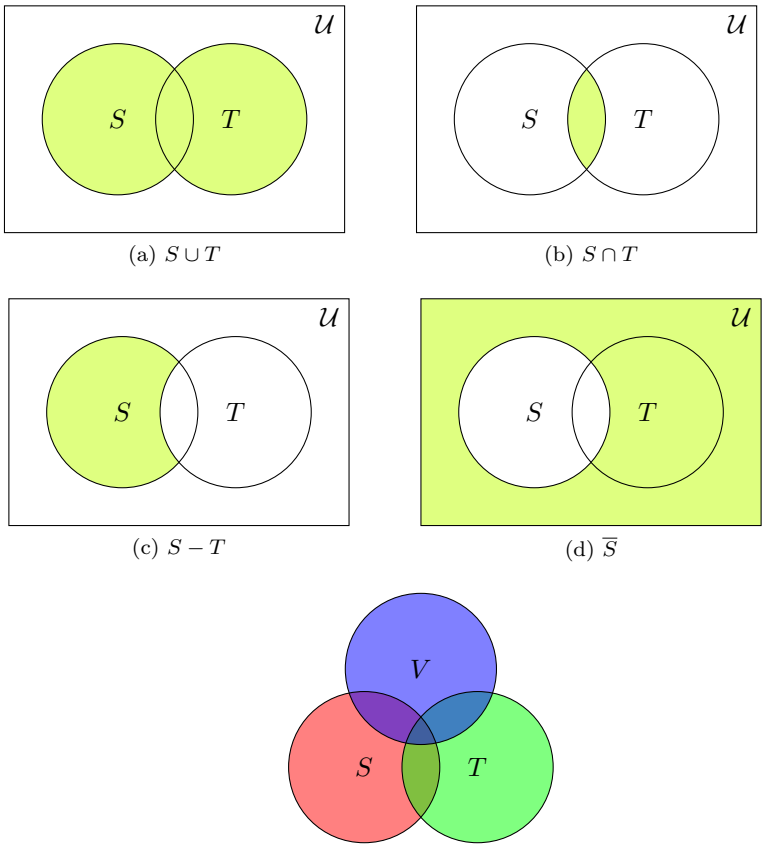
Example 1.7.

- $\{1, 2\} \subseteq \{1, 2, 3\}$.
- $\{1, 2\} \subset \{1, 2, 3\}$.
- $\{1, 2, 3\} \subseteq \{1, 2, 3\}$.
- $\{1, 2, 3\} \not\subset \{1, 2, 3\}$.
- For any set S, $\emptyset \subseteq S$.
- For every set $S \neq \emptyset$, $\emptyset \subset S$.
- $S \subseteq T$ and $T \subseteq S$ if and only if $S = T$.

Finally, it is time to formalize operations on sets. Given two collection of objects, we may want to merge the collections (set union), identify the objects in common (set intersection), or identify the objects unique to one collection (set difference). We may also be interested in knowing all possible ways of picking one object from each collection (Cartesian product), or all possible ways of picking some objects from just one of the collections (power set).

Definition 1.8 (Set operations)**.** Given sets S and T, we define the following operations:

- *Power Sets.* $\mathcal{P}(S)$ is the set of all subsets of S.
- *Cartesian Product.* $S \times T = \{(s, t) \mid s \in S, t \in T\}$.
- *Union.* $S \cup T = \{x \mid x \in S \text{ or } x \in T\}$, set of elements in S or T.
- *Intersection.* $S \cap T = \{x \mid x \in S, x \in T\}$, set of elements in S and T.
- *Difference.* $S - T = \{x \mid x \in S, x \notin T\}$, set of elements in S but not T.
- *Complements.* $\overline{S} = \{x \mid x \notin S\}$, set of elements not in S. This is only meaningful when we have an implicit universe $\mathcal{U}$ of objects, i.e., $\overline{S} = \{x \mid x \in \mathcal{U}, x \notin S\}$.



(a) $S \cup T$

(b) $S \cap T$

(c) $S - T$

(d) $\overline{S}$

(e) Venn diagram with three sets.

Figure 1.1: Venn diagrams of sets S, T, and V under universe $\mathcal{U}$.

Example 1.9. Let $S = \{1, 2, 3\}$, $T = \{3, 4\}$, $V = \{a, b\}$. Then:

- $\mathcal{P}(T) = \{\emptyset, \{3\}, \{4\}, \{3, 4\}\}$.
- $S \times V = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$.
- $S \cup T = \{1, 2, 3, 4\}$.
- $S \cap T = \{3\}$.
- $S - T = \{1, 2\}$.
- If we are dealing with the set of all integers, $\overline{S} = \{\ldots, -2, -1, 0, 4, 5, \ldots\}$.

Some set operations can be visualized using Venn diagrams. See Figure 1.1. To give an example of working with these set operations, consider the following set identity.

Theorem 1.10. *For all sets S and T, $S = (S \cap T) \cup (S - T)$.*

Proof. We can visualize the set identity using Venn diagrams (see Figure 1.1b and 1.1c). To formally prove the identity, we will show both of the following:

$$S \subseteq (S \cap T) \cup (S - T) \tag{1.1}$$

$$(S \cap T) \cup (S - T) \subseteq S \tag{1.2}$$

To prove (1.1), consider any element $x \in S$. Either $x \in T$ or $x \notin T$.

- If $x \in T$, then $x \in S \cap T$, and thus also $x \in (S \cap T) \cup (S - T)$.
- If $x \notin T$, then $x \in (S - T)$, and thus again $x \in (S \cap T) \cup (S - T)$.

To prove (1.2), consider any $x \in (S \cap T) \cup (S - T)$. Either $x \in S \cap T$ or $x \in S - T$

- If $x \in S \cap T$, then $x \in S$
- If $x \in S - T$, then $x \in S$. ■

In computer science, we frequently use the following additional notation (these notation can be viewed as short hands):

Definition 1.11. Given a set S and a natural number $n \in \mathbb{N}$,

- S^n is the set of length n "strings" (equivalently n-tuples) with alphabet S. Formally we define it as the product of n copies of S (i.e., $S \times S \times \cdots \times S$).
- S^* is the set of finite length "strings" with alphabet S. Formally we define it as the union of $S^0 \cup S^1 \cup S^2 \cup \cdots$, where S^0 is a set that contains only one element: the empty string (or the empty tuple "()").
- $[n]$ is the set $\{0, 1, \ldots, n-1\}$.

Commonly seen set includes $\{0,1\}^n$ as the set of n-bit strings, and $\{0,1\}^*$ as the set of finite length bit strings. Also observe that $|[n]| = n$.

Before we end this section, let us revisit our informal definition of sets: an unordered "collection" of objects. In 1901, Russel came up with the following "set", known as Russel's paradox[1]:

$$S = \{x \mid x \notin x\}$$

That is, S is the set of all sets that don't contain themselves as an element. This might seem like a natural "collection", but is $S \in S$? It's not hard to see that $S \in S \leftrightarrow S \notin S$. The conclusion today is that S is not a good "collection" of objects; it is not a set.

So how will know if $\{x \mid x$ satisfies some condition$\}$ is a set? Formally, sets can be defined axiomatically, where only collections constructed from a careful list of rules are considered sets. This is outside the scope of this course. We will take a short cut, and restrict our attention to a well-behaved universe. Let E

[1] A folklore version of this paradox concerns itself with barbers. Suppose in a town, the only barber shaves all and only those men in town who do not shave themselves. This seems perfectly reasonable, until we ask: Does the barber shave himself?

be all the objects that we are interested in (numbers, letters, etc.), and let $\mathcal{U} = E \cup \mathcal{P}(E) \cup \mathcal{P}(\mathcal{P}(E))$, i.e., E, subsets of E and subsets of subsets of E. In fact, we may extend $\mathcal{U}$ with three power set operations, or indeed any *finite* number of power set operations. Then, $S = \{x \mid x \in \mathcal{U}$ and some condition holds$\}$ is always a set.

1.2 Relations

Definition 1.12 (Relations)**.** A relation on sets S and T is a subset of $S \times T$. A relation on a single set S is a subset of $S \times S$.

Example 1.13. "Taller-than" is a relation on people; $(A, B) \in$ "Taller-than" if person A is taller than person B. "$\geq$" is a relation on $\mathbb{R}$; "$\geq$"$= \{(x, y) \mid x, y \in \mathbb{R},\ x \geq y\}$.

Definition 1.14 (Reflexitivity, symmetry, and transitivity)**.** A relation R on set S is:

- *Reflexive* if $(x, x) \in R$ for all $x \in S$.
- *Symmetric* if whenever $(x, y) \in R$, $(y, x) \in R$.
- *Transitive* if whenever $(x, y), (y, z) \in R$, then $(x, z) \in R$

Example 1.15.

- "$\leq$" is reflexive, but "$<$" is not.
- "sibling-of" is symmetric, but "$\leq$" and "sister-of" is not.
- "sibling-of", "$\leq$", and "$<$" are all transitive, but "parent-of" is not ("ancestor-of" is transitive, however).

Definition 1.16 (Graph of relations)**.** The graph of a relation R over S is an directed graph with nodes corresponding to elements of S. There is an edge from node x to y if and only if $(x, y) \in R$. See Figure 1.2.

Theorem 1.17. *Let R be a relation over S.*

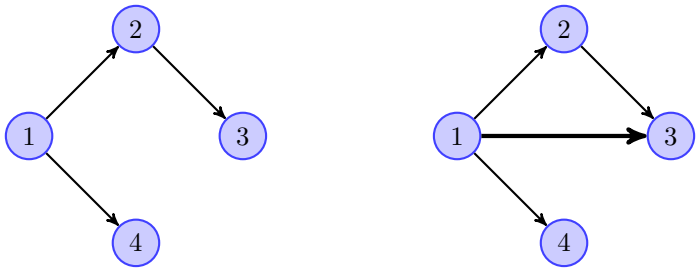
- *R is reflexive iff its graph has a self-loop on every node.*
- *R is symmetric iff in its graph, every edge goes both ways.*
- *R is transitive iff in its graph, for any three nodes x, y and z such that there is an edge from x to y and from y to z, there exist an edge from x to z.*
- *More naturally, R is transitive iff in its graph, whenever there is a path from node x to node y, there is also a direct edge from x to y.*

Proof. The proofs of the first three parts follow directly from the definitions. The proof of the last bullet relies on induction; we will revisit it later. ■

Definition 1.18 (Transitive closure)**.** The transitive closure of a relation R is the least (i.e., smallest) transitive relation R^* such that $R \subseteq R^*$.

Pictorially, R^* is the connectivity relation: if there is a path from x to y in the graph of R, then $(x, y) \in R^*$.

Example 1.19. Let $R = \{(1,2),(2,3),(1,4)\}$ be a relation (say on set $\mathbb{Z}$). Then $(1,3) \in R^*$ (since $(1,2),(2,3) \in R$), but $(2,4) \notin R^*$. See Figure 1.2.



(a) The relation $R = \{(1,2),(2,3),(1,4)\}$ (b) The relation R^*, transitive closure of R

Figure 1.2: The graph of a relation and its transitive closure.

Theorem 1.20. *A relation R is transitive iff $R = R^*$.*

Definition 1.21 (Equivalence relations)**.** A relation R on set S is an equivalence relation if it is reflexive, symmetric and transitive.

Equivalence relations capture the every day notion of "being the same" or "equal".

Example 1.22. The following are equivalence relations:

- Equality, "=", a relation on numbers (say $\mathbb{N}$ or $\mathbb{R}$).
- Parity $= \{(x, y) \mid x, y$ are both even or both odd$\}$, a relation on integers.

1.3 Functions

Definition 1.23. A function $f : S \to T$ is a "mapping" from elements in set S to elements in set T. Formally, f is a relation on S and T such that for each $s \in S$, there exists a unique $t \in T$ such that $(s, t) \in R$. S is the *domain* of f, and T is the *range* of f. $\{y \mid y = f(x)$ for some $x \in S\}$ is the *image* of f.

We often think of a function as being characterized by an algebraic formula, e.g., $y = 3x - 2$ characterizes the function $f(x) = 3x - 2$. Not all formulas characterizes a function, e.g. $x^2 + y^2 = 1$ is a relation (a circle) that is not

a function (no unique y for each x). Some functions are also not easily characterized by an algebraic expression, e.g., the function mapping past dates to recorded weather.

Definition 1.24 (Injection)**.** $f : S \to T$ is injective (one-to-one) if for every $t \in T$, there exists at most one $s \in S$ such that $f(s) = t$, Equivalently, f is injective if whenever $s \neq s$, we have $f(s) \neq f(s)$.

Example 1.25.

- $f : \mathbb{N} \to \mathbb{N}$, $f(x) = 2x$ is injective.
- $f : \mathbb{R}^+ \to \mathbb{R}^+$, $f(x) = x^2$ is injective.
- $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$ is not injective since $(-x)^2 = x^2$.

Definition 1.26 (Surjection)**.** $f : S \to T$ is surjective (onto) if the image of f equals its range. Equivalently, for every $t \in T$, there exists some $s \in S$ such that $f(s) = t$.

Example 1.27.

- $f : \mathbb{N} \to \mathbb{N}$, $f(x) = 2x$ is not surjective.
- $f : \mathbb{R}^+ \to \mathbb{R}^+$, $f(x) = x^2$ is surjective.
- $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$ is not injective since negative reals don't have real square roots.

Definition 1.28 (Bijection)**.** $f : S \to T$ is bijective, or a *one-to-one correspondence*, if it is injective and surjective.

See Figure 1.3 for an illustration of injections, surjections, and bijections.

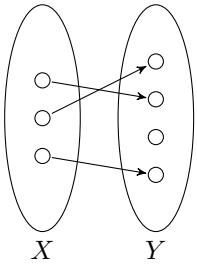
Definition 1.29 (Inverse relation)**.** Given a function $f : S \to T$, the inverse relation f^{-1} on T and S is defined by $(t, s) \in f^{-1}$ if and only if $f(s) = t$.

If f is bijective, then f^{-1} is a function (unique inverse for each t). Similarly, if f is injective, then f^{-1} is a also function if we restrict the domain of f^{-1} to be the image of f. Often an easy way to show that a function is one-to-one is to exhibit such an inverse mapping. In both these cases, $f^{-1}(f(x)) = x$.

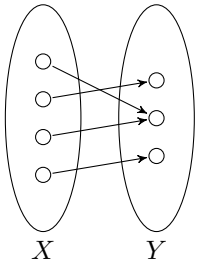
1.4 Set Cardinality, revisited

Bijections are very useful for showing that two sets have the same number of elements. If $f : S \to T$ is a bijection and S and T are finite sets, then $|S| = |T|$. In fact, we will extend this definition to infinite sets as well.

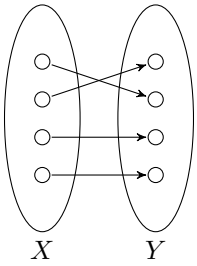
Definition 1.30 (Set cardinality)**.** Let S and T be two potentially infinite sets. S and T have the same cardinality, written as $|S| = |T|$, if there exists a bijection $f : S \to T$ (equivalently, if there exists a bijection $f' : T \to S$). T has cardinality at larger or equal to S, written as $|S| \leq |T|$, if there exists an injection $g : S \to T$ (equivalently, if there exists a *surjection* $g' : T \to S$).



(a) An injective function from X to Y.



(b) A surjective function from X to Y.



(c) A bijective function from X to Y.

Figure 1.3: Injective, surjective and bijective functions.

To "intuitively justify" Definition 1.30, see Figure 1.3. The next theorem shows that this definition of cardinality corresponds well with our intuition for size: if both sets are at least as large as the other, then they have the same cardinality.

Theorem 1.31 (Cantor-Bernstein-Schroeder)**.** *If $|S| \leq |T|$ and $|T| \leq |S|$, then $|S| = |T|$. In other words, given injective maps, $g : S \to T$ and $h : T \to S$, we can construct a bijection $f : S \to T$.*

We omit the proof of Theorem 1.31; interested readers can easily find multiple flavours of proofs online. Set cardinality is much more interesting when the sets are infinite. The cardinality of the natural numbers is extra special, since you can "count" the numbers. (It is also the "smallest infinite set", a notion that is outside the scope of this course.)

Definition 1.32. A set S is countable if it is finite or has the same cardinality as $\mathbb{N}^+$. Equivalently, S is countable if $|S| \leq |\mathbb{N}^+|$.

Example 1.33.

- $\{1, 2, 3\}$ is countable because it is finite.
- $\mathbb{N}$ is countable because it has the same cardinality as $\mathbb{N}^+$; consider $f : \mathbb{N}^+ \to \mathbb{N}$, $f(x) = x - 1$.

- The set of positive even numbers, $S = \{2, 4, \dots\}$, is countable consider $f : \mathbb{N}^+ \to S$, $f(x) = 2x$.

Theorem 1.34. *The set of positive rational numbers $\mathbb{Q}^+$ are countable.*

Proof. $\mathbb{Q}^+$ is clearly not finite, so we need a way to count $\mathbb{Q}^+$. Note that double counting, triple counting, even counting some element infinite many times is okay, as long as we eventually count *all* of $\mathbb{Q}^+$. I.e., we implicitly construct a surjection $f : \mathbb{N}^+ \to \mathbb{Q}^+$.

Let us count in the following way. We first order the rational numbers p/q by the value of $p + q$; then we break ties by ordering according to p. The ordering then looks like this:

- First group ($p + q = 2$): $1/1$
- Second group ($p + q = 3$): $1/2$, $2/1$
- Third group ($p + q = 4$): $1/3$, $2/2$, $3/1$

Implicitly, we have $f(1) = 1/1$, $f(2) = 1/2$, $f(3) = 2/1$, etc. Clearly, f is a surjection. See Figure 1.4 for an illustration of f. ■

1/1	1/2	1/3	1/4	1/5 ⋯
2/1	2/2	2/3	2/4	2/5 ⋯
3/1	3/2	3/3	3/4	3/5 ⋯
4/1	4/2	4/3	4/4	4/5 ⋯
5/1	5/2	5/3	5/4	5/5 ⋯
⋮	⋮	⋮	⋮	⋮

Figure 1.4: An infinite table containing all positive rational numbers (with repetition). The red arrow represents how f traverses this table—how we count the rationals.

Theorem 1.35. *There exists sets that are not countable.*

Proof. Here we use Cantor's diagonlization argument. Let S be the set of infinite sequences $(d_1, d_2, \dots)$ over digits $\{0, 1\}$. Clearly S is infinite. To show that there cannot be a bijection with $\mathbb{N}^+$, we proceed by contradiction. Suppose $f : \mathbb{N}^+ \to S$ is a bijection. We can then enumerate these strings using f, producing a 2-dimensional table of digits:

$$f(1) = s^1 = (d_1^1, d_2^1, d_3^1, \dots)$$
$$f(2) = s^2 = (d_1^2, d_2^2, d_3^2, \dots)$$
$$f(3) = s^3 = (d_1^3, d_2^3, d_3^3, \dots)$$

Now consider $s^* = (1-d_1^1, 1-d_2^2, 1-d_3^3, \dots)$, i.e., we are taking the diagonal of the above table, and flipping all the digits. Then for any n, s^* is different from s^n in the n^{th} digit. This contradicts the fact that f is a bijection. ■

Theorem 1.36. *The real interval* $[0, 1]$ *(the set of real numbers between 0 and 1, inclusive) is uncountable.*

Proof. We will show that $|[0, 1]| \geq |S|$, where S is the same set as in the proof of Theorem 1.35. Treat each $s = (d_1, d_2, \dots) \in S$ as the real number between 0 and 1 with the binary expansion $0.d_1d_2\cdots$. Note that this does not establish a bijection; some real numbers have two binary expansions, e.g., $0.1 = 0.0111\cdots$ (similarly, in decimal expansion, we have $0.1 = 0.0999\cdots$[2]).

We may overcome this "annoyance" in two ways:

- Since each real number can have at most two decimal representations (most only have one), we can easily extend the above argument to show that $|S| \leq |[0, 2]|$ (i.e., map $[0, 1]$ to one representation, and $[1, 2]$ to the other). It remains to show that $|[0, 1]| = |[0, 2]|$ (can you think of a bijection here?).
- We may repeat Cantor's diagonlization argument as in the proof of Theorem 1.35, in decimal expansion. When we construct s^*, avoid using the digits 9 and 0 (e.g., use only the digits 4 and 5). ■

A major open problem in mathematics (it was one of Hilbert's 23 famous problems listed 1900) was whether there exists some set whose cardinality is between $\mathbb{N}$ and $\mathbb{R}$ (can you show that $\mathbb{R}$ has the same cardinality as $[0, 1]$?).

Here is a naive candidate: $\mathcal{P}(\mathbb{N})$. Unfortunately, $\mathcal{P}(\mathbb{N})$ has the same cardinality as $[0, 1]$. Note that every element $S \in \mathcal{P}(\mathbb{N})$ corresponds to an infinitely long sequence over digits $\{0, 1\}$ (the n^{th} digit is 1 if and only if the number $n \in S$). Again, we arrive at the set S in the proof of Theorem 1.35.

The Continuum Hypothesis states that no such set exists. Gödel and Cohen together showed (in 1940 and 1963) that this can neither be proved nor disproved using the standard axioms underlying mathematics (we will talk more about axioms when we get to logic).

[2]For a proof, consider letting $x = 0.0999\cdots$, and observe that $10x - x = 0.999\cdots - 0.0999\cdots = 0.9$, which solves to $x = 0.1$.

Chapter 2

Proofs and Induction

"Pics or it didn't happen."
– the internet

There are many forms of mathematical proofs. In this chapter we introduce several basic types of proofs, with special emphasis on a technique called *induction* that is invaluable to the study of discrete math.

2.1 Basic Proof Techniques

In this section we consider the following general task: given a premise X, how do we show that a conclusion Y holds? One way is to give a **direct proof**. Start with premise X, and directly deduce Y through a series of logical steps. See Claim 2.1 for an example.

Claim 2.1. *Let n be an integer. If n is even, then n^2 is even. If n is odd, then n^2 is odd.*

Direct proof. If n is even, then $n = 2k$ for an integer k, and

$$n^2 = (2k)^2 = 4k^2 = 2 \cdot (2k^2), \text{ which is even.}$$

If n is odd, then $n = 2k + 1$ for an integer k, and

$$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2 \cdot (2k^2 + 2k) + 1, \text{ which is odd.} \quad \blacksquare$$

There are also several forms of indirect proofs. A **proof by contrapositive** starts by assuming that the conclusion Y is false, and deduce that the premise X must also be false through a series of logical steps. See Claim 2.2 for an example.

Claim 2.2. *Let n be an integer. If n^2 is even, then n is even.*

Proof by contrapositive. Suppose that n is not even. Then by Claim 2.1, n^2 is not even as well. (Yes, the proof ends here.) $\blacksquare$

A **proof by contradiction**, on the other hand, assumes both that the premise X is true and the conclusion Y is false, and reach a logical fallacy. We give another proof of Claim 2.2 as example.

Proof by contradiction. Suppose that n^2 is even, but n is odd. Applying Claim 2.1, we see that n^2 must be odd. But n^2 cannot be both odd and even! ■

In their simplest forms, it may seems that a direct proof, a proof by contrapositive, and a proof and contradiction may just be restatements of each other; indeed, one can always phrase a direct proof or a proof by contrapositive as a proof by contradiction (can you see how?). In more complicated proofs, however, choosing the "right" proof technique sometimes simplify or improve the aesthetics of a proof. Below is an interesting use of proof by contradiction.

Theorem 2.3. *$\sqrt{2}$ is irrational.*

Proof by contradiction. Assume for contradiction that $\sqrt{2}$ is rational. Then there exists integers p and q, with no common divisors, such that $\sqrt{2} = p/q$ (i.e., the reduced fraction). Squaring both sides, we have:

$$2 = \frac{p^2}{q^2} \qquad \Rightarrow \qquad 2q^2 = p^2$$

This means p^2 is even, and by Claim 2.2 p is even as well. Let us replace p by $2k$. The expression becomes:

$$2q^2 = (2k)^2 = 4k^2 \qquad \Rightarrow \qquad q^2 = 2k^2$$

This time, we conclude that q^2 is even, and so q is even as well. But this leads to a contradiction, since p and q now shares a common factor of 2. ■

We end the section with the (simplest form of the) AM-GM inequality.

Theorem 2.4 (Simple AM-GM inequality)**.** *Let x and y be non-negative reals. Then,*

$$\frac{x+y}{2} \geq \sqrt{xy}$$

Proof by contradiction. Assume for contradiction that

$$\begin{aligned}
& \frac{x+y}{2} < \sqrt{xy} \\
\Rightarrow \quad & \frac{1}{4}(x+y)^2 < xy && \text{squaring non-negative values} \\
\Rightarrow \quad & x^2 + 2xy + y^2 < 4xy \\
\Rightarrow \quad & x^2 - 2xy + y^2 < 0 \\
\Rightarrow \quad & (x-y)^2 < 0
\end{aligned}$$

But this is a contradiction since squares are always non-negative. ■

Note that the proof Theorem 2.4 can be easily turned into a direct proof; the proof of Theorem 2.3, on the other hand, cannot.

2.2 Proof by Cases and Examples

Sometimes the easiest way to prove a theorem is to split it into several cases.

Claim 2.5. *$(n+1)^2 \geq 2^n$ for all integers n satisfying $0 \leq n \leq 5$.*

Proof by cases. There are only 6 different values of n. Let's try them all:

n	$(n+1)^2$		2^n
0	1	$\geq$	1
1	4	$\geq$	2
2	9	$\geq$	4
3	16	$\geq$	8
4	25	$\geq$	16
5	36	$\geq$	32

■

Claim 2.6. *For all real x, $|x^2| = |x|^2$.*

Proof by cases. Split into two cases: $x \geq 0$ and $x < 0$.

- If $x \geq 0$, then $|x^2| = x^2 = |x|^2$.
- If $x < 0$, then $|x^2| = x^2 = (-x)^2 = |x|^2$.

■

When presenting a proof by cases, make sure that *all* cases are covered! For some theorems, we only need to construct one case that satisfy the theorem statement.

Claim 2.7. *Show that there exists some n such that $(n+1)^2 \geq 2^n$.*

Proof by example. $n = 6$. ■

Sometimes we find a counterexample to disprove a theorem.

Claim 2.8. *Prove or disprove that $(n+1)^2 \geq 2^n$ for all $n \in \mathbb{N}$.*

Proof by (counter)example. We choose to disprove the statement. Check out $n = 6$. Done. ■

The next proof does not explicitly construct the example asked by the theorem, but proves that such an example exists anyways. These type of proofs (among others) are *non-constructive*.

Theorem 2.9. *There exists irrational numbers x and y such that x^y is rational.*

Non-constructive proof of existence. We know $\sqrt{2}$ is irrational from Theorem 2.3. Let $z = \sqrt{2}^{\sqrt{2}}$.

- If z is rational, then we are done ($x = y = \sqrt{2}$).
- If z is irrational, then take $x = z = \sqrt{2}^{\sqrt{2}}$, and $y = \sqrt{2}$. Then:

$$x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\sqrt{2}} = \sqrt{2}^2 = 2$$

is indeed a rational number. ■

Here is another non-constructive existence proof. The game of *Chomp* is a 2-player game played on a "chocolate bar" made up of a rectangular grid. The players take turns to choose one block and "eat it" (remove from the board), together all other blocks that are below it or to its right (the whole lower right quadrant). The top left block is "poisoned" and the player who eats this loses.

Theorem 2.10. *Suppose the game of Chomp is played with rectangular grid strictly larger than* 1×1*. Player 1 (the first player) has a winning strategy.*

Proof. Consider following first move for player 1: eat the lower right most block. We have two cases[1]:

- Case 1: There is a winning strategy for player 1 starting with this move. In this case we are done.
- Case 2: There is no winning strategy for player 1 starting with this move. In this case there is a winning strategy for player 2 following this move. But this winning strategy for player 2 is also a valid winning strategy for players 1, since the next move made by player 2 can be mimicked by player 1 (here we need the fact that the game is symmetric between the players). ■

While we have just shown that Player 1 can always win in a game of Chomp, no constructive strategy for Player 1 has been found for general rectangular grids (i.e., you cannot buy a strategy guide in store that tells you how to win Chomp). For a few specific cases though, we do know good strategies for Player 1. E.g., given a $n \times n$ square grid, Player 1 starts by removing a $n-1 \times n-1$ (unique) block, leaving an L-shaped piece of chocolate with two "arms"; thereafter, Player 1 simply mirrors Player 2's move, i.e., whenever Player 2 takes a bite from one of the arms, Player 1 takes the same bite on the other arm.

A our last example, consider tilling a 8×8 chess board with dominoes (2×1 pieces), i.e., the whole board should be covered by dominoes without any dominoes over lapping each other or sticking out.

Q: Can we tile it?
A: Yes. Easy to give a proof by example (constructive existence proof).

[1] Here we use the well-known fact of 2-player, deterministic, finite-move games without ties: any move is either a winning move (i.e., there is a strategy following this move that forces a win), or allows the opponent to follow up with a winning move. See Theorem 2.14 later for a proof of this fact.

Q: What if I remove one grid of the check board?
A: No. Each domino covers 2 grids, so the number of covered grids is always even, but the board has 63 pieces (direct proof / proof by contradiction).

Q: What if I remove the top left and bottom right grids?
A: No. Each domino covers 1 grid of each colors. The top left and bottom right grids have the same color, however, so the remaining board has more white grids than black (or more black grids than white) (direct proof / proof by contradiction).

2.3 Induction

We start with the most basic form of induction: induction over the natural numbers. Suppose we want to show that a statement is true for all natural numbers, e.g., for all n, $1 + 2 + \cdots + n = n(n + 1)/2$. The basic idea is to approach the proof in two steps:

1. First prove that the statement is true for $n = 1$. This is called the **base case**.

2. Next prove that whenever the statement is true for case n, then it is also true for case $n + 1$. This is called the **inductive step**.

The base case shows that the statement is true for $n = 1$. Then, by repeatedly applying the inductive step, we see that the statement is true for $n = 2$, and then $n = 3$, and then $n = 4, 5, \ldots$; we just covered all the natural numbers! Think of pushing over a long line of dominoes. The induction step is just like setting up the dominoes; we make sure that if a domino falls, so will the next one. The base case is then analogous to pushing down the first domino. The result? All the dominoes fall.

Follow these steps to write an inductive proof:

1. Start by formulating the **inductive hypothesis** (i.e., what you want to prove). It should be parametrized by a natural number. E.g., $P(n)$: $1 + 2 + \cdots + n = n(n + 1)/2$.

2. Show that $P(base)$ is true for some appropriate base case. Usually $base$ is 0 or 1.

3. Show that the inductive step is true, i.e., assume $P(n)$ holds and prove that $P(n + 1)$ holds as well.

Violà, we have just shown that $P(n)$ holds for all $n \geq base$. Note that the base case does not always have to be 0 or 1; we can start by showing that something is $P(n)$ is true for $n = 5$; this combined with the inductive step shows that $P(n)$ is true for all $n \geq 5$. Let's put our new found power of inductive proofs to the test!

Claim 2.11. *For all positive integers n, $1 + 2 + \cdots + n = n(n+1)/2$.*

Proof. Define out induction hypothesis $P(n)$ to be true if

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

Base case: $P(1)$ is clearly true by inspection.

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well:

$$\begin{aligned}
\sum_{i=1}^{n+1} i &= \left(\sum_{i=1}^{n} i \right) + (n+1) \\
&= \frac{1}{2} n(n+1) + n + 1 && \text{using } P(n) \\
&= \frac{1}{2} \left(n(n+1) + 2(n+1) \right) = \frac{1}{2} \left((n+1)(n+2) \right)
\end{aligned}$$

This is exactly $P(n+1)$. ■

Claim 2.12. *For any finite set S, $|\mathcal{P}(S)| = 2^{|}S|$.*

Proof. Define our induction hypothesis $P(n)$ to be true if for every finite set S of cardinality $|S| = n$, $|\mathcal{P}(S)| = 2^n$.

Base case: $P(0)$ is true since the only finite set of size 0 is the empty set $\emptyset$, and the power set of the empty set, $\mathcal{P}(\emptyset) = \{\emptyset\}$, has cardinality 1.

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. Consider a finite set S of cardinality $n+1$. Pick an element $e \in S$, and consider $S' = S - \{e\}$. By the induction hypothesis, $|\mathcal{P}(S')| = 2^n$.

Now consider $\mathcal{P}(S)$. Observe that a set in $\mathcal{P}(S)$ either contains e or not; furthermore, there is a one-to-one correspondence between the sets containing e and the sets not containing e (can you think of the bijection?). We have just partitioned $\mathcal{P}(S)$ into two equal cardinality subsets, one of which is $\mathcal{P}(S')$. Therefore $|\mathcal{P}(S)| = 2|\mathcal{P}(S)| = 2^{n+1}$. ■

Claim 2.13. *The following two properties of graphs are equivalent (recall that these are the definitions of transitivity on the graph of a relation):*

1. *For any three nodes x, y and z such that there is an edge from x to y and from y to z, there exist an edge from x to z.*
2. *Whenever there is a path from node x to node y, there is also a direct edge from x to y.*

Proof. Clearly property 2 implies property 1. We use induction to show that property 1 implies property 2 as well. Let G be a graph on which property 1 holds. Define our induction hypothesis $P(n)$ to be true if for every path of length n in G from node x to node y, there exists a direct edge from x to y.

Base case: $P(1)$ is simply true (path of length 1 is already a direct edge).

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. Consider a path of length $n+1$ from node x to node y, and let z be the first node after x on the path. We now have a path of length n from node z to y, and by the induction hypothesis, a direct edge from z to y. Now that we have a directly edge from x to z and from z to y, property 1 implies that there is a direct edge from x to y. ■

Theorem 2.14. *In a deterministic, finite 2-player game of perfect information without ties, either player 1 or player 2 has a winning strategy, i.e., a strategy that guarantees a win.*[2,3]

Proof. Let $P(n)$ be the theorem statement for n-move games.

Base case: $P(1)$ is trivially true. Since only player 1 gets to move, if there exists some move that makes player 1 win, then player 1 has a winning strategy; otherwise player 2 always wins and has a winning strategy (the strategy of doing nothing).

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. Consider some $n+1$-move game. After player 1 makes the first move, we end up in a n-move game. Each such game has a winning strategy for either player 1 or player 2 by $P(n)$.

- If all these games have a winning strategy for player 2[4], then no matter what move player 1 plays, player 2 has a winning strategy
- If one these games have a winning strategy for player 1, then player 1 has a winning strategy (by making the corresponding first move). ■

In the next example, induction is used to prove only a subset of the theorem to give us a jump start; the theorem can then be completed using other techniques.

Theorem 2.15 (AM-GM Inequality)**.** *Let $x_1, x_2, \ldots, x_n$ be a sequence of non-negative reals. Then*

$$\frac{1}{n}\sum_i x_i \geq \left(\prod_i x_i\right)^{1/n}$$

[3]By deterministic, we mean the game has no randomness and depends on only on player moves (e.g., not backgammon). By finite, we mean the game is always ends in some predetermined fix number of moves; in chess, even though there are infinite sequences of moves that avoid both checkmates and stalemates, many draw rules (e.g., cannot have more than 100 consecutive moves without captures or pawn moves) ensures that chess is a finite game. By perfect information, we mean that both players knows each other's past moves (e.g., no fog of war).

[4] By this we mean the player 1 of the n-move game (the next player to move) has a winning strategy

Proof. In this proof we use the notation

$$\mathrm{AM}(x_1,\ldots,x_n)=\frac{1}{n}\sum_{i=1}^{n}x_i \qquad \mathrm{GM}(x_1,\ldots,x_n)=\left(\prod_{i=1}^{n}x_i\right)^{1/n}$$

Let us first prove the AM-GM inequality for values of $n=2^k$. Define our induction hypothesis $P(k)$ to be true if AM-GM holds for $n=2^k$.

Base case: $P(0)$ (i.e., $n=1$) trivially holds, and $P(1)$ (i.e., $n=2$) was shown in Theorem 2.4.

Inductive Step: Assume $P(k)$ is true; we wish to show that $P(k+1)$ is true as well. Given a sequence of length 2^{k+1}, $\vec{X}=(x_1,..,x_{2^{k+1}})$, we split it into two sequences $\vec{X}_1=(x_1,\ldots,x_2^k)$, $\vec{X}_2=(x_{2^k+1},x_{2^k+2},\ldots,x_{2^{k+1}})$. Then:

$$\begin{aligned}
\mathrm{AM}(\vec{X}) &= \frac{1}{2}(\mathrm{AM}(\vec{X}_1)+\mathrm{AM}(\vec{X}_2)) \\
&\geq \frac{1}{2}(\mathrm{GM}(\vec{X}_1)+\mathrm{GM}(\vec{X}_2)) && \text{by the induction hypothesis } P(k) \\
&= \mathrm{AM}(\mathrm{GM}(\vec{X}_1),\mathrm{GM}(\vec{X}_2)) \\
&\geq \mathrm{GM}(\mathrm{GM}(\vec{X}_1),\mathrm{GM}(\vec{X}_2)) && \text{by Theorem 2.4, i.e., } P(1) \\
&= \left(\left(\prod_{i=1}^{2^k}x_i\right)^{\frac{1}{2^k}}\left(\prod_{i=2^k+1}^{2^{k+1}}x_i\right)^{\frac{1}{2^k}}\right)^{1/2} \\
&= \left(\prod_{i=1}^{2^{k+1}}x_i\right)^{\frac{1}{2^{k+1}}} = \mathrm{GM}(\vec{X})
\end{aligned}$$

We are now ready to show the AM-GM inequality for sequences of all lengths. Given a sequence $\vec{X}=(x_1,\ldots,x_n)$ where n is not a power of 2, find the smallest k such that $2^k>n$. Let $\alpha=\mathrm{AM}(\vec{X})$, and consider a new sequence

$$\vec{X}'=(x_1,\ldots,x_n,x_{n+1}=\alpha,x_{n+2}=\alpha,\ldots,x_{2^k}=\alpha)$$

and verify that $\mathrm{AM}(\vec{X}')=\mathrm{AM}(\vec{X})=\alpha$. Apply $P(k)$ (the AM-GM inequality for sequences of length 2^k), we have:

$$\begin{aligned}
& & \mathrm{AM}(\vec{X}') = \alpha \geq \mathrm{GM}(\vec{X}') &= \left(\prod_{i=1}^{2^k} x_i\right)^{1/2^k} \\
\Rightarrow & & \alpha^{2^k} &\geq \prod_{i=1}^{2^k} x_i = \prod_{i=1}^{n} x_i \cdot \alpha^{2^k - n} \\
\Rightarrow & & \alpha^n &\geq \prod_{i=1}^{n} x_i \\
\Rightarrow & & \alpha &\geq \left(\prod_{i=1}^{n} x_i\right)^{1/n} = \mathrm{GM}(\vec{X})
\end{aligned}$$

This finishes our proof (recalling that $\alpha = \mathrm{AM}(\vec{X})$). ■

Note that for the inductive proof in Theorem 2.15, we needed to show both base cases $P(0)$ and $P(1)$ to avoid circular arguments, since the inductive step relies on $P(1)$ to be true.

A common technique in inductive proofs is to define a *stronger* induction hypothesis than is needed by the theorem. A stronger induction hypothesis $P(n)$ sometimes make the induction step simpler, since we would start each induction step with a stronger premise. As an example, consider the game of "coins on the table". The game is played on a round table between two players. The players take turns putting on one penny at a time onto the table, without overlapping with previous pennies; the first player who cannot add another coin losses.

Theorem 2.16. *The first player has a winning strategy in the game of "coins on the table".*

Proof. Consider the following strategy for player 1 (the first player). Start first by putting a penny centered on the table, and in all subsequent moves, simply mirror player 2's last move (i.e., place a penny diagonally opposite of player 2's last penny). We prove by induction that player 1 can always put down a coin, and therefore will win eventually (when the table runs out of space).

Define the induction hypothesis $P(n)$ to be true if on the n^{th} move of player 1, player 1 can put down a penny according to its strategy, and leave the table symmetric about the centre (i.e., looks the same if rotated 180 degrees).

Base case: $P(1)$ holds since player 1 can always start by putting one penny at the centre of the table, leaving the table symmetric.

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. By the induction hypothesis, after player 1's n^{th} move, the table is symmetric. Therefore, if player 2 now puts down a penny, the diagonally

opposite spot must be free of pennies, allowing player 1 to set down a penny as well. Moreover, after player 1's move, the table is back to being symmetric. ■

The Towers of Hanoi is a puzzle game where there is threes poles, and a number of increasingly larger rings that are originally all stacked in order of size on the first pole, largest at the bottom. The goal of the puzzle is to move all the rings to another pole (pole 2 or pole 3), with the rule that:

- You may only move one ring a time, and it must be the top most ring in one of the three potential stacks.
- At any point, no ring may be placed on top of a smaller ring.[5]

Theorem 2.17. *The Towers of Hanoi with n rings can be solved in $2^n - 1$ moves.*

Proof. Define the induction hypothesis $P(n)$ to be true if the theorem statement is true for n rings.

Base case: $P(1)$ is clearly true. Just move the ring.

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. Number the rings 1 to $n+1$, from smallest to largest (top to bottom on the original stack). First move rings 1 to n from pole 1 to pole 2; this takes $2^n - 1$ steps by the induction hypothesis $P(n)$. Now move ring $n+1$ from pole 1 to pole 3. Finally, move rings 1 to n from pole 2 to pole 3; again, this takes $2^n - 1$ steps by the induction hypothesis $P(n)$. In total we have used $(2^n - 1) + 1 + (2^n - 1) = 2^{n+1} - 1$ moves. (Convince yourself that this recursive definition of moves will never violate the rule that no ring may be placed on top of a smaller ring.) ■

Legends say that such a puzzle was found in a temple with $n = 64$ rings, left for the priests to solve. With our solution, that would require $2^{64} - 1 \approx 1.8 \times 10^{19}$ moves. Is our solution just silly and takes too many moves?

Theorem 2.18. *The Towers of Hanoi with n rings requires at least $2^n - 1$ moves to solve. Good luck priests!*

Proof. Define the induction hypothesis $P(n)$ to be true if the theorem statement is true for n rings.

Base case: $P(1)$ is clearly true. You need to move the ring.

Inductive Step: Assume $P(n)$ is true; we wish to show that $P(n+1)$ is true as well. Again we number the rings 1 to $n+1$, from smallest to largest (top to bottom on the original stack). Consider ring $n+1$. It needs to be moved at some point. Without loss of generality, assume its final destination is pole 3. Let the k^{th} move be the first move where ring $n+1$ is moved away from pole 1 (to pole 2 or 3), and let the k'^{th} move be the last move where ring $n+1$ is moved to pole 3 (away from pole 1 to pole 2),

[5] Squashing small rings with large rings is bad, m'kay?

Before performing move k, all n other rings must first be moved to the remaining free pole (pole 3 or 2); by the induction hypothesis $P(n)$, $2^n - 1$ steps are required before move k. Similarly, after performing move k, all n other rings must be on the remaining free pole (pole 2 or 1); by the induction hypothesis $P(n)$, $2^n - 1$ steps are required after move k' to complete the puzzle. In the best case where $k = k'$ (i.e., they are the same move), we still need at least $(2^n - 1) + 1 + (2^n - 1) = 2^{n+1} - 1$ moves. ■

Strong Induction

Taking the dominoes analogy one step further, a large domino may require the combined weight of all the previous toppling over before it topples over as well. The mathematical equivalent of this idea is *strong induction*. To prove that a statement $P(n)$ is true for (a subset of) positive integers, the basic idea is:

1. First prove that $P(n)$ is true for some base values of n (e.g., $n = 1$). These are the **base cases**.

2. Next prove that if $P(k)$ is true for $1 \leq k \leq n$, then $P(n+1)$ is true. This is called the **inductive step**.

How many base cases do we need? It roughly depends on the following factors:

- What is the theorem? Just like basic induction, if we only need $P(n)$ to be true for $n \geq 5$, then we don't need base cases $n < 5$.

- What does the induction hypothesis need? Often to show $P(n + 1)$, instead of requiring that $P(k)$ be true for $1 \leq k \leq n$, we actually need, say $P(n)$ and $P(n - 1)$ to be true. Then having the base case of $P(1)$ isn't enough for the induction hypothesis to prove $P(3)$; $P(2)$ is another required base case.

Let us illustrate both factors with an example.

Claim 2.19. *Suppose we have an unlimited supply of 3 cent and 5 cent coins. Then we can pay any amount ≥ 8 cents.*

Proof. Let $P(n)$ be the true if we can indeed form n cents with 3 cent and 5 cent coins.

Base case: $P(8)$ is true since $3 + 5 = 8$.

Inductive Step: Assume $P(k)$ is true for $8 \leq k \leq n$; we wish to show that $P(n+1)$ is true as well. This seems easy; if $P(n-2)$ is true, then adding another 3 cent coin gives us $P(n + 1)$. But the induction hypothesis doesn't necessarily say $P(n - 2)$ is true! For $(n + 1) \geq 11$, the induction hypothesis does apply (since $n - 2 \geq 8$). For $n + 1 = 9$ or 10, we have to do more work.

Additional base cases: $P(9)$ is true since $3 + 3 + 3 = 9$, and $P(10)$ is true since $5 + 5 = 10$. ■

With any induction, especially strong induction, it is *very important* to check for sufficient base cases! Here is what might happen in a faulty strong inductive proof.[6] Let $P(n)$ be true if for all groups of n women, whenever one women is blonde, then all of the women are blonde; since there is at least one blonde in the world, once I am done with the proof, every women in the world will be blonde!

Base case: $P(1)$ is clearly true.

Induction step: Suppose $P(k)$ is true for all $1 \leq k \leq n$; we wish to show $P(n+1)$ is true as well. Given a set W of $n+1$ women in which $x \in W$ is blonde, take any two strict subsets $A, B \subsetneq W$ (in particular $|A|, |B| < n+1$) such that they both contain the blonde ($x \in A, x \in B$), and $A \cup B = W$ (no one is left out). Applying the induction hypothesis to A and B, we conclude that all the women in A and B are blonde, and so everyone in W is blonde.

What went wrong?[7]

2.4 Inductive Definitions

In addition to being a proof technique, induction can be used to *define* mathematical objects. Some basic examples include products or sums of sequences:

- The factorial function $n!$ over non-negative integers can be formally defined by
$$0! = 1; \qquad (n+1)! = n! \cdot (n+1)$$
- The cumulative sum of a sequence $x_1, \ldots, x_k$, often written as $S(n) = \sum_{i=1}^{n} x_i$, can be formally defined by
$$S(0) = 0; \qquad S(n+1)! = S(n) + x_{n+1}$$

Just like inductive proofs, inductive definitions start with a "base case" (e.g., defining $0! = 1$), and has an "inductive step" to define the rest of the values (e.g., knowing $0! = 1$, we can compute $1! = 1 \cdot 1 = 1$, $2! = 1 \cdot 2 = 2$, and so on).

Recurrence Relations

When an inductive definition generates a sequence (e.g., the factorial sequence is $1, 1, 2, 6, 24, \ldots$), we call the definition a *recurrence relation*. We can generalize inductive definitions and recurrence relations in a way much like we

[6] Another example is to revisit Claim 2.19. If we use the same proof to show that $P(n)$ is true for all $n \geq 3$, without the additional base cases, the proof will be "seemingly correct". What is the obvious contradiction?

[7] Hint: Can you trace the argument when $n = 2$?

generalize inductive proofs with strong induction. For example, consider a sequence defined by:

$$a_0 = 1; \qquad a_1 = 2; \qquad a_n = 4a_{n-1} - 4a_{n-2}$$

According to the definition, the next few terms in the sequence will be

$$a_2 = 4; \qquad a_3 = 8$$

At this point, the sequence looks suspiciously as if $a_n = 2^n$. Let's prove this by induction!

Proof. Define $P(n)$ to be true if $a_n = 2^n$.

Base case: $P(0)$ and $P(1)$ are true since $a_0 = 1 = 2^0, a_1 = 2 = 2^1$.

Inductive Step: Assume $P(k)$ is true for $0 \leq k \leq n$; we wish to show that $P(n+1)$ is true as well for $n + 1 \geq 2$. We have

$$\begin{aligned} a_{n+1} &= 4a_n - 4a_{n-1} \\ &= 4 \cdot 2^n - 4 \cdot 2^{n-1} && \text{by } P(n) \text{ and } P(n-1) \\ &= 2^{n+2} - 2^{n+1} = 2^{n+1} \end{aligned}$$

This is exactly $P(n+1)$. ■

Remember that it is very important to check the *all* the base cases (especially since this proof uses strong induction). Let us consider another example:

$$b_0 = 1; \qquad b_1 = 1; \qquad b_n = 4b_{n-1} - 3b_{n-2}$$

From the recurrence part of the definition, its looks like the sequence $(b_n)_n$ will eventually out grow the sequence $(a_n)_n$. Based only on this intuition, let us conjecture that $b_n = 3^n$.

Possibly correct proof. Define $P(n)$ to be true if $b_n = 3^n$.

Base case: $P(0)$ is true since $b_0 = 1 = 3^0$.

Inductive Step: Assume $P(k)$ is true for $0 \leq k \leq n$; we wish to show that $P(n+1)$ is true as well for $n + 1 \geq 3$. We have

$$\begin{aligned} b_{n+1} &= 4b_n - 3b_{n-1} \\ &= 4 \cdot 3^n - 3 \cdot 3^{n-1} && \text{by } P(n) \text{ and } P(n-1) \\ &= (3^{n+1} + 3^n) - 3^n = 3^{n+1} \end{aligned}$$

■

Wow! Was that a lucky guess or what. Let us actually compute a few terms of $(b_n)_n$ to make sure...

$$\begin{aligned} b_2 &= 4b_1 - 3b_0 = 4 - 3 = 1, \\ b_3 &= 4b_2 - 3b_1 = 4 - 3 = 1, \\ &\vdots \qquad ☹ \end{aligned}$$

Looks like in fact, $b_n = 1$ for all n (as an exercise, prove this by induction). What went wrong with our earlier "proof"? Note that $P(n-1)$ is only well defined if $n \geq 1$, so the inductive step does not work when we try to show $P(1)$ (when $n = 0$). As a result we need an extra base case to handle $P(1)$; a simple check shows that it is just not true: $b_1 = 1 \neq 3^1 = 3$. (On the other hand, if we define $b'_0 = 1$, $b'_1 = 3$, and $b'_n = 4b'_{n-1} - 3b'_{n-2}$, then we can recycle our "faulty proof" and show that $b'_n = 3^n$).

In the examples so far, we guessed at a closed form formula for the sequences $(a_n)_n$ and $(b_n)_n$, and then proved that our guesses were correct using induction. For certain recurrence relations, there are direct methods for computing a closed form formula of the sequence.

Theorem 2.20. *Consider the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ with $c_2 \neq 0$, and arbitrary base cases for a_0 and a_1. Suppose that the polynomial $x^2 - (c_1 x + c_2)$ has two distinct roots r_1 and r_2 (these roots are non-zero since $c_2 \neq 0$). Then there exists constants α and β such that $a_n = \alpha x_1^n + \beta x_2^n$.*

Proof. The polynomial $f(x) = x^2 - (c_1 x + c_2)$ is called the *characteristic polynomial* for the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$. Its significance can be explained by the sequence $(r^0, r^1, \dots)$ where r is a root of $f(x)$; we claim that this sequence satisfies the recurrence relation (with base cases set as r^0 and r^1). Let $P(n)$ be true if $a_n = r^n$.

Inductive Step: Assume $P(k)$ is true for $0 \leq k \leq n$; we wish to show that $P(n+1)$ is true as well. Observe that:

$$\begin{aligned} a_{n+1} &= c_1 a_n + c_2 a_{n-1} \\ &= c_1 r^n + c_2 r^{n-1} && \text{by } P(n-1) \text{ and } P(n) \\ &= r^{n-1}(c_1 r + c_2) \\ &= r^{n-1} \cdot r^2 && \text{since } r \text{ is a root of } f(x) \\ &= r^{n+1} \end{aligned}$$

Recall that there are two distinct roots, r_1 and r_2, so we actually have two sequences that satisfy the recurrence relation (under proper base cases). In fact, because the recurrence relation is *linear* (a_n depends linearly on a_{n-1} and a_{n-2}), and *homogeneous* (there is no constant term in the recurrence relation), any sequence of the form $a_n = \alpha r_1^n + \beta r_2^n$ will satisfy the recurrence relation; (this can be shown using a similar inductive step as above).

Finally, does sequences of the form $a_n = \alpha r_1^n + \beta r_2^n$ cover all possible base cases? The answer is yes. Given any base case $a_0 = a_0^*$, $a_1 = a_1^*$, we can solve for the unique value of α and β using the linear system:

$$\begin{aligned} a_0^* &= \alpha r_1^0 + \beta r_2^0 = \alpha + \beta \\ a_1^* &= \alpha r_1^1 + \beta r_2^1 = \alpha r_1 + \beta r_2 \end{aligned}$$

The studious reader should check that this linear system always has a unique solution (say, by checking that the determinant of the system is non-zero). ∎

The technique outlined in Theorem 2.20 can be extended to any recurrence relation of the form

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

for some constant k; the solution is always a linear combination of k sequences of the form $(r^0, r^1, r^2, \dots)$, one for each distinct root r of the characteristic polynomial

$$f(x) = x^k - (c_1 x^{k-1} + c_2 x^{k-2} + \cdots + c_k)$$

In the case that $f(x)$ has duplicate roots, say when a root r has multiplicity m, in order to still have a total of k distinct sequences, we associate the following m sequences with r:

$$\begin{array}{llllll}
(& r^0, & r^1, & r^2, & \dots, & r^n, \quad \dots) \\
(& 0 \cdot r^0, & 1 \cdot r^1, & 2 \cdot r^2, & \dots, & nr^n, \quad \dots) \\
(& 0^2 \cdot r^0, & 1^2 \cdot r^1, & 2^2 \cdot r^2, & \dots, & n^2 r^n, \quad \dots) \\
 & & & \vdots & & \\
(& 0^{m-1} \cdot r^0, & 1^{m-1} \cdot r^1, & 2^{m-1} \cdot r^2, & \dots, & n^{m-1} r^n, \quad \dots)
\end{array}$$

For example, if $f(x)$ has degree 2 and has a unique root r with multiplicity 2, then the general form solution to the recurrence is

$$a_n = \alpha r^n + \beta n r^n$$

We omit the proof of this general construction. Interestingly, the same technique is used in many other branches of mathematics (for example, to solve linear ordinary differential equations).

As an example, let us derive a closed form expression to the famous Fibonacci numbers.

Theorem 2.21. *Define the Fibonacci sequence inductively as*

$$f_0 = 0; \quad f_1 = 1; \quad f_n = f_{n-1} + f_{n-2}$$

Then

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n \tag{2.1}$$

Proof. It is probably hard to guess (2.1); we will derive it from scratch. The characteristic polynomial here is $f(x) = x^2 - (x+1)$, which has roots

$$\frac{1+\sqrt{5}}{2}, \qquad \frac{1-\sqrt{5}}{2}$$

This means the Fibonacci sequence can be expressed as

$$f_n = \alpha \left(\frac{1+\sqrt{5}}{2} \right)^n + \beta \left(\frac{1-\sqrt{5}}{2} \right)^n$$

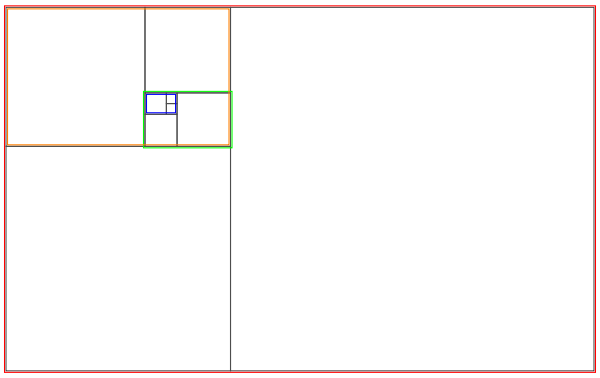


Figure 2.1: Approximating the golden ratio with rectangles whose side lengths are consecutive elements of the Fibonacci sequence. Do the larger rectangles look more pleasing than the smaller rectangles to you?

for constants α and β. Substituting $f_0 = 0$ and $f_1 = 1$ gives us

$$\begin{aligned} 0 &= \alpha + \beta \\ 1 &= \alpha\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right) \end{aligned}$$

which solves to $\alpha = 1/\sqrt{5}$, $\beta = -1/\sqrt{5}$. ■

As a consequence of (2.1), we know that for large n,

$$f_n \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

because the other term approaches zero. This in term implies that

$$\lim_{n\to\infty} \frac{f_{n+1}}{f_n} = \frac{1+\sqrt{5}}{2}$$

which is the **golden ratio**. It is widely believed that a rectangle whose ratio (length divided by width) is golden is pleasing to the eye; as a result, the golden ratio can be found in many artworks and architectures throughout history.

2.5 Fun Tidbits

We end the section with a collection of fun examples and anecdotes on induction.

Induction and Philosophy: the Sorites Paradox

The sorites paradox, stated below, seems to question to validity of inductive arguments:

Base Case: One grain of sand is not a heap of sand.

Inductive Step: If n grains of sand is not a heap of sand, then $n + 1$ grains of sand is not a heap of sand either.

We then conclude that a googol (10^{100}) grains of sand is not a heap of sand (this is more than the number of atoms in the observable universe by some estimates). What went wrong? The base case and the inductive step is perfectly valid! There are many "solutions" to this paradox, one of which is to blame it on the *vagueness* of the word "heap"; the notion of vagueness is itself a topic of interest in philosophy.

Induction and Rationality: the Traveller's Dilemma

Two travelers, Alice and Bob, fly with identical vases; the vases get broken. The airline company offers to reimburse Alice and Bob in the following way. Alice and Bob, separately, is asked to quote the value of the vase at between 2 to 100 dollars. If they come up with the same value, then the airline will reimburse both Alice and Bob at the agreed price. If they come up with different values, m and m' with $m < m'$, then the person who quoted the smaller amount m gets $m + 2$ dollars in reimbursement, while the person who quoted the bigger amount m' gets $m - 2$ dollars. What should they do?

Quoting \$100 seems like a good strategy. But if Alice knows Bob will quote \$100, then Alice should quote \$99. In fact, quoting quoting 99 is sometimes better and never worse than quoting \$100. We conclude that it is never "rational" to quote \$100.

But now that Alice and Bob knows that the other person will never quote \$100, quoting \$98 is now a better strategy than quoting \$99. We conclude that it is never "rational" to quote \$99 or above.

We can continue the argument by induction (this argument is called backward induction in the economics literature) that the only rational thing for Alice and Bob to quote is \$2. Would you quote \$2 (and do you think you are "rational")?

Induction and Knowledge: the Muddy Children

Suppose a group of children are in a room, and some have mud on their forehead. All the children can see everyone else's forehead, but not their won (no mirrors, and no touching), and so they do not know if they themselves are muddy. The father comes into the room and announce that some of the children are muddy, and asks if anyone knows (for sure) that they are the ones who are muddy. Everyone says no. The father then asks the same question again, but everyone still says no. The father keeps asking the same question over and

over again, until all of a sudden, all the muddy children in the room simultaneously says yes, that they do know they are muddy. How many children said yes? How many rounds of questioning has there been?

Claim 2.22. *All the muddy children says yes in the n^{th} round of questioning (and not earlier) if and only if there are n muddy children.*

Proof Sketch. Since we have not formally defined the framework of knowledge, we are constrained to an informal proof sketch. Let $P(n)$ be true if the claim is true for n muddy children.

Base case: We start by showing $P(1)$. If there is only one child that is muddy, the child sees that everyone else is clean, and can immediately deduces that he/she must be muddy (in order for there to be someone muddy in the room). On the other hand, if there are 2 or more muddy children, then all the muddy children see at least another muddy child, and cannot tell apart whether "some kids are muddy" refer to them or the other muddy children in the room.

Inductive Step: Assume $P(k)$ is true for $0 \leq k \leq n$; we wish to show $P(n+1)$. Suppose there are exactly $n+1$ muddy children. Since there are more than n muddy children, it follows by the induction hypothesis that that no one will speak before round $n+1$. From the view of the muddy children, they see n other muddy kids, and know from the start that there are either n or $n+1$ muddy children in total (depending on whether they themselves are muddy). But, by the induction hypothesis, they know that if there were n muddy children, then someone would have said yes in round n; since no one has said anything yet, each muddy child deduces that he/she is indeed muddy and says yes in round $n+1$. Now suppose there are strictly more than $n+1$ muddy children. In this case, everyone sees at least $n+1$ muddy children already. By the induction hypothesis, every children *knows* from the beginning that that no one will speak up in the first n round. Thus in $n+1^{\text{st}}$ round, they have no more information about who is muddy than when the father first asked the question, and thus they cannot say yes. ■

Induction Beyond the Natural Numbers [Optional Material]

In this chapter we have restricted our study of induction to the natural numbers. Our induction hypotheses (e.g., $P(n)$) are always parametrized by a natural number, and our inductive definitions (e.g., the Fibonacci sequence) have always produced a sequence of objects indexed by the natural numbers. Can we do induction over some other set that is not the natural numbers? Clearly we can do induction on, say, all the even natural numbers, but what about something more exotic, say the rational numbers, or say the set of C programs?

Rational Numbers. Let us start with an ill-fated example of induction on the rational numbers. We are going to prove (incorrectly) that all non-negative

rational numbers q, written in reduced form a/b, must be even in the numerator (a) and odd in the denominator (b). Let $P(q)$ be true if the claim is true for rational number q.

Base Case: $P(0)$ is true since $0 = 0/1$ in its reduced form.

Inductive Step: Suppose $P(k)$ is true for all rationals $0 \leq k < n$. We wish to show that $P(n)$ is true as well. Consider the rational number $n/2$ and let a'/b' be its reduced form. By the induction hypothesis $P(n/2)$, a' is even and b' is odd. It follows that n, in its reduced form, is $(2a')/b$, and thus $P(n)$ is true.

Looking carefully at the proof, we are not making the same mistakes as before in our examples for strong induction: to show $P(n)$, we rely only on $P(n/2)$, which *always* satisfies $0 \leq n/2 < n$, so we are not simply missing base cases. The only conclusion is that induction just "does not make sense" for the rational numbers.

C programs. On the other hand, we *can* inductively define and reason about C programs. Let us focus on a simpler example: the set of (very limited) arithmetic expressions defined by the following context free grammar:

$$expr \rightarrow 0 \mid 1 \mid (expr + expr) \mid (expr \times expr)$$

We can interpret this context free grammar as an inductive definition of arithmetic expressions:

Base Case: An arithmetic can be the digit 0 or 1.

Inductive (Recursive) Definition: An arithmetic expression can be of the form "$(expr_1 + expr_2)$" or "$(expr_1 \times expr_2)$", where $expr_1$ and $expr_2$ are itself arithmetic expressions.

Notice that this inductive definition does not give us a sequence of arithmetic expressions! We can also define the value of an arithmetic expression inductively:

Base Case: The arithmetic expression "0" has value 0, and the expression "1" has value 1.

Inductive Definition: An arithmetic expression of the form "$(expr_1 + expr_2)$" has value equal to the *sum* of the values of $expr_1$ and $expr_2$. Similarly, an arithmetic expression of the form "$(expr_1 \times expr_2)$" has value equal to the *product* of the values of $expr_1$ and $expr_2$.

We can even use induction to prove, for example, that any expression of length n must have value $\leq 2^{2^n}$.

So how are natural numbers, rational numbers and C programs different from one another? To make it more bewildering, did we not show a mapping between the rational numbers and the natural numbers? The answer lies in the way we induct through these sets or, metaphorically speaking, how the

dominoes are lined up. The formal desired property on lining up the dominoes is called **well-founded relations**, and is beyond the scope of this course. Instead, here is a thought experiment that illustrates the difference in the inductive procedure between the numerous correct inductive proofs in this chapter, and the (faulty) inductive proof on the rational numbers. Suppose we want to verify that $1 + 2 + \cdots + 10 = 10 \cdot 11/2 = 55$; this is shown in Claim 2.11, our very first inductive argument. Here is how we may proceed, knowing the inductive proof:

- We verify the inductive step, and conclude that if $1+2+\cdots+9 = 9\cdot 10/2$ is true (the induction hypothesis), then $1 + 2 + \cdots + 10 = 10 \cdot 11/2$ is true. It remains to verify that $1 + 2 + \cdots + 9 = 9 \cdot 10/2$.
- To verify that $1 + 2 + \cdots + 9 = 9 \cdot 10/2$, we again look at the inductive step and conclude that it remains to verify that $1 + 2 + \cdots + 8 = 8 \cdot 9/2$.
- Eventually, after a *finite* number of steps (9 steps in this case), it remains to verify the that $1 = 1$, which is shown in the base case of the induction.

Similarly, to verify that "$((1+(0\times 1))\times(1+1))$" is a valid arithmetic expression, we first verify that it is of the form "$(expr_1 \times expr_2)$", and recursively verify that "$(1 + (0 \times 1))$" and "$(1 + 1)$" are valid arithmetic expressions. Again, this recursive verification will end in *finite* time.

Finally, let us consider our faulty example with rational numbers. To show that the number $2/3$ in reduced form is an even number over an odd number, we need to check the claim for the number $1/3$, and for that we need to check $1/6$, and $1/12$, and $\ldots$; this never ends, so we never have a complete proof of the desired (faulty) fact.

Chapter 3

Number Theory

"Mathematics is the queen of sciences and number theory is the queen of mathematics."
– Carl Friedrich Gauss

Number theory is the study of numbers (in particular the integers), and is one of the purest branch of mathematics. Regardless, it has many applications in computer science, particularly in cryptography, the underlying tools that build modern services such as secure e-commerce. In this chapter, we will touch on the very basics of number theory, and put an emphasis on its applications to cryptography.

3.1 Divisibility

A fundamental relation between two numbers is whether or not one divides another.

Definition 3.1 (Divisibility)**.** Let $a, b \in \mathbb{Z}$ with $a \neq 0$. We say that a divides b, denoted by $a|b$, if there exists some $k \in Z$ such that $b = ak$.

Example 3.2. $3|9$, $5|10$, but $3 \nmid 7$.

The following theorem lists a few well-known properties of divisibility.

Theorem 3.3. *Let $a, b, c \in \mathbb{Z}$.*

1. *If $a|b$ and $a|c$ then $a|(b + c)$*
2. *If $a|b$ then $a|bc$*
3. *If $a|b$ and $b|c$ then $a|c$ (i.e., transitivity).*

Proof. We show only item 1; the other proofs are similar (HW). By definition,

$$a|b \Rightarrow \text{there exist } k_1 \in Z \text{ such that } b = k_1 a$$

$$a|c \Rightarrow \text{there exist } k_2 \in Z \text{ such that } c = k_2 a$$

Therefore $b + c = k_1 a + k_2 a = (k_1 + k_2)a$, so $a|(b + c)$. ■

Corollary 3.4. *Let $a, b, c \in \mathbb{Z}$. If $a|b$ and $a|c$, then $a|mb+nc$ for any $m, n \in \mathbb{Z}$.*

We learn in elementary school that even when integers don't divide evenly, we can compute the quotient and the remainder.

Theorem 3.5 (Division Algorithm)**.** *For any $a \in \mathbb{Z}$ and $d \in \mathbb{N}^+$, there exist unique $q, r \in \mathbb{Z}$ s.t. $a = dq + r$ and $0 \leq r < d$.*

q is called the quotient *and denoted by $q = a$ div d.*

r is called the remainder *and denoted by $r = a \bmod d$.*

For example, dividing 99 by 10 gives a quotient of $q = 99$ div $10 = 9$ and remainder of $r = 99 \bmod 10 = 9$, satisfying $99 = 10(9) + 9 = 10q + r$. On the other hand, dividing 99 by 9 gives a quotient of $q = 99$ div $9 = 11$ and remainder of $r = 99 \bmod 9 = 0$. Again, we have $99 = 11(9) + 0 = 11q + r$. Onwards to proving the theorem.

Proof. Given $a \in \mathbb{Z}$ and $d \in \mathbb{N}^+$, let $q = \lfloor a/d \rfloor$ (the greatest integer $\leq a/d$), and let $r = a - dq$. By choice of q and r, we have $a = dq + r$. We also have $0 \leq r < d$, because q is the *largest* integer such that $dq \leq a$. It remains to show uniqueness.

Let $q', r' \in Z$ be any other pairs of integers satisfying $a = dq' + r'$ and $0 \leq r' < d$. We would have:

$$\begin{aligned} & dq + r = dq' + r' \\ \Rightarrow \qquad & d \cdot (q - q') = r' - r. \end{aligned}$$

This implies that $d|(r'-r)$. But $-(d-1) \leq r'-r \leq d-1$ (because $0 \leq r, r' < d$), and the only number divisible by d between $-(d-1)$ and $d-1$ is 0. Therefore we must have $r' = r$, which in turn implies that $q' = q$. ■

Greatest Common Divisor

Definition 3.6 (Greatest Common Divisor)**.** Let $a, b \in \mathbb{Z}$ with a, b not both 0. The greatest common divisor of a and b, denoted by $\gcd(a, b)$, is the largest integer d such that $d|a$ and $d|b$.

Example 3.7.

$$\gcd(4, 12) = \gcd(12, 4) = \gcd(-4, -12) = \gcd(-12, 4) = 4$$

$$\gcd(12, 15) = 3 \qquad \gcd(3, 5) = 1 \qquad \gcd(20, 0) = 20$$

Euclid designed one of the first known algorithms in history (for any problem) to compute the greatest common divisor:

Algorithm 1 EuclidAlg(a, b), $a, b, \in \mathbb{N}^+$, a, b not both 0

```
if b = 0 then
  return a;
else
  return EuclidAlg(b, a mod b);
end if
```

Example 3.8. Let's trace Euclid's algorithm on inputs 414 and 662.

$$\begin{aligned}\textsc{EuclidAlg}(414, 662) &\to \textsc{EuclidAlg}(662, 414) \to \textsc{EuclidAlg}(414, 248)\\ &\to \textsc{EuclidAlg}(248{,}166) \to \textsc{EuclidAlg}(166, 82) \to \textsc{EuclidAlg}(82, 2)\\ &\to \textsc{EuclidAlg}(2, 0) \to 2\end{aligned}$$

The work for each step is shown below:

$$\begin{aligned}662 &= 414(1) + 248\\ 414 &= 248(1) + 166\\ 248 &= 166(1) + 82\\ 166 &= 82(2) + 2\\ 82 &= 41(2) + 0\end{aligned}$$

We now prove that Euclid's algorithm is correct in two steps. First, we show that if the algorithm terminates, then it does output the correct greatest common divisor. Next, we show that Euclid's algorithm always terminates (and does so rather quickly).

Lemma 3.9. *Let $a, b \in N$, $b \neq 0$. Then* $\gcd(a, b) = \gcd(b, a \bmod b)$.

Proof. It is enough to show that the common divisors of a and b are the same as the common divisors of b and $(a \bmod b)$. If so, then the two pairs of numbers must also share the same *greatest* common divisor.

By the division algorithm, there exist unique $q, r \in \mathbb{Z}$ such that $a = bq + r$ and $0 \leq r < b$. Also recall that by definition, $r = a \bmod b = a - bq$. Let d be a common divisor of a and b, i.e., d divides both a and b. Then d also divides $r = abq$ (by Corollary 3.4). Thus d is a common divisor of b and r. Similarly, let d' be a common divisor of b and r. Then d also divides $a = bq + r$. Thus d is a common divisor of a and b. ■

Theorem 3.10. *Euclid's algorithm (*EuclidAlg*) produces the correct output if it terminates.*

Proof. This can be shown by induction, using Lemma 3.9 as the inductive step. (What would be the base case?) ■

We now show that Euclid's algorithm always terminates.

Claim 3.11. *For every two recursive calls made by* EuclidAlg*, the first argument a is halved.*

Proof. Suppose EuclidAlg(a, b) is called. If $b \leq a/2$, then in the next recursive call EuclidAlg$(b, a \bmod b)$ already has the property that the first argument is halved. Otherwise, we have $b > a/2$, so $a \bmod b \leq a/2$. Then after two recursive calls (first EuclidAlg$(b, a \bmod b)$, then EuclidAlg$(a \bmod b, b \bmod (a \bmod b))$), we have the property that the first argument is halved. ■

The next theorem follows directly from Claim 3.11.

Theorem 3.12. *Euclid's algorithm, on input* EuclidAlg(a, b) *for $a, b \in \mathbb{N}^+$, a, b not both 0, always terminates. Moreover it terminates in time proportional to $\log_2 a$.*

We end the section with a useful fact on greatest common divisors.

Theorem 3.13. *Let $a, b \in N^+$, a, b not both 0. Then, there exist $s, t \in \mathbb{Z}$ such that $sa + tb = \gcd(a, b)$.*

Theorem 3.13 shows that we can give a *certificate* for the greatest common divisor. From Corollary 3.4, we already know that any common divisor of a and b also divides $sa+tb$. Thus, if we can identify a common divisor d of a and b, and show that $d = sa + tb$ for some s and t, this demonstrates d is in fact the *greatest* common divisor ($d = \gcd(a, b)$). And there is more good news! This certificate can be produced by slightly modifying Euclid's algorithm (often called the extended Euclid's algorithm); this also constitutes as a constructive proof of Theorem 3.13. We omit the proof here and give an example instead.

Example 3.14. Suppose we want to find $s, t \in \mathbb{Z}$ such that $s(252) + t(198) = \gcd(252, 198) = 18$. Run Euclid's algorithm, but write out the equation $a = bq + r$ for each recursive call of EuclidAlg.

$$\begin{aligned}
&\text{EuclidAlg}(252, 198) && 252 = 1(198) + 54 && (3.1)\\
\rightarrow&\text{EuclidAlg}(198, 54) && 198 = 3(54) + 36 && (3.2)\\
\rightarrow&\text{EuclidAlg}(54, 36) && 54 = 1(36) + 18 && (3.3)\\
\rightarrow&\text{EuclidAlg}(36, 18) && 36 = 2(18) + 0 && (3.4)
\end{aligned}$$

We can construct s and t by substituting the above equations "backwards":

$$\begin{aligned}
18 &= 1(54) - 1(36) && \text{by (3.3)}\\
&= 1(54) - (1(198) - 3(54)) && \text{by (3.2)}\\
&= 4(54) - 1(198)\\
&= 4(252 - 1(198)) - 1(198) && \text{by (3.1)}\\
&= 4(252) - 5(198)
\end{aligned}$$

We conclude that $\gcd(252, 198) = 18 = 4(252) - 5(198)$.

3.2 Modular Arithmetic

Modular arithmetic, as the name implies, is arithmetic on the remainders of integers, with respect to a *fixed* divisor. A central idea to modular arithmetic is congruences: two integers are considered "the same" if they have the same remainder with respect to the fixed divisor.

Definition 3.15. Let $a, b \in \mathbb{Z}$, $m \in \mathbb{N}^+$. We say that a and b are congruent modular m, denoted by $a \equiv b \pmod{m}$, if $m|(a-b)$ (i.e., if there exists $k \in \mathbb{Z}$ such that $a - b = km$).

As a direct consequence, we have $a \equiv a \pmod{m}$ for any $m \in \mathbb{N}^+$.

Claim 3.16. *$a \equiv b \pmod{m}$ if and only if a and b have the same remainder when divided by m, i.e., $a \bmod m = b \bmod m$.*

Proof. We start with the if direction. Assume a and b have the same remainder when divided by m. That is, $a = q_1 m + r$ and $b = q_2 m + r$. Then we have

$$a - b = (q_1 - q_2)m \quad \Rightarrow \quad m|(a-b)$$

For the only if direction, we start by assuming $m|(a-b)$. Using the division algorithm, let $a = q_1 m + r_1$, $b = q_2 m + r_2$ with $0 \leq r_1, r_2 < m$. Because $m|(a-b)$, we have

$$m|(q_1 m + r_1 - (q_2 m + r_2))$$

Since m clearly divides $q_1 m$ and $q_2 m$, it follows by Corollary 3.4 that

$$m|r_1 - r_2$$

But $-(m-1) \leq r_1 - r_2 \leq m - 1$, so we must have $a \bmod m = r_1 = r_2 = b \bmod m$. ∎

The next theorem shows that addition and multiplication "carry over" to the modular world (specifically, addition and multiplication can be computed before or after computing the remainder).

Theorem 3.17. *If $a \equiv b \pmod{m}$, and $c \equiv d \pmod{m}$ then*

1. *$a + c \equiv b + d \pmod{m}$*
2. *$ac \equiv bd \pmod{m}$*

Proof. For item 1, we have

$$\begin{aligned}
& a \equiv b \pmod{m} \text{ and } c \equiv d \pmod{m} \\
\Rightarrow\ & m|(a-b) \text{ and } m|(c-d) \\
\Rightarrow\ & m|((a-b)+(c-d)) && \text{by Corollary 3.4} \\
\Rightarrow\ & m|((a+c)-(b+d)) \\
\Rightarrow\ & a + c \equiv b + d \pmod{m}
\end{aligned}$$

For item 2, using Claim 3.16, we have unique integers r and r' such that

$$a = q_1 m + r \qquad b = q_2 m + r$$
$$c = q_1' m + r' \qquad d = q_2' m + r'$$

This shows that

$$ac = q_1 m \cdot q_1' m + q_1 m r' + q_1' m r + r r'$$
$$bd = q_2 m \cdot q_2' m + q_2 m r' + q_2' m r + r r'$$

which clearly implies that $m|(ac - bd)$. ■

Clever usage of Theorem 3.17 can simplify many modular arithmetic calculations.

Example 3.18.

$$11^{999} \equiv 1^{999} \equiv 1 \pmod{10}$$
$$9^{999} \equiv (-1)^{999} \equiv -1 \equiv 9 \pmod{10}$$
$$7^{999} \equiv 49^{499} \cdot 7 \equiv (-1)^{499} \cdot 7 \equiv -7 \equiv 3 \pmod{10}$$

Note that exponentiation was not included in Theorem 3.17. Because multiplication does carry over, we have $a^e \equiv (a \bmod n)^e \pmod{n}$; we have already used this fact in the example. However, in general we cannot perform modular operations on the exponent first, i.e., $a^e \not\equiv a^{e \bmod n} \pmod{n}$.

Applications of Modular Arithmetic

In this section we list some applications of modular arithmetic, and, as we promised, give an example of an application to cryptography.

Hashing. The age-old setting that call for hashing is simple. How do we efficiently retrieve (store/delete) a large number of records? Take for example student records, where each record has a unique 10-digit student ID. We cannot (or do not want) a table of size 10^{10} to index all the student records indexed by their ID. The solution? Store the records in an array of size N where N is a bit bigger than the expected number of students. The record for student ID is then stored in position $h(\text{ID})$ where h is a *hash function* that maps IDs to $\{1, \ldots, N\}$. One very simple hash-function would be

$$h(k) = k \bmod N$$

ISBN. Most published books today have a 10 or 13-digit ISBN number; we will focus on the 10-digit version here. The ISBN identifies the country of publication, the publisher, and other useful data, but all these information are stored in the first 9 digits; the 10^{th}digit is a redundancy check for errors.

The actual technical implementation is done using modular arithmetic. Let $a_1, \ldots, a_{10}$ be the digits of an ISBN number. In order to be a valid ISBN number, it must pass the check:

$$a_1 + 2a_2 + ..9a_9 + 10a_{10} \equiv 0 \pmod{11}$$

This test would detect an error if:

- a single digit was changed, or
- a transposition occurred, i.e., two digits were swapped (this is why in the check, we multiply a_i by i).

If 2 or more errors occur, the errors may cancel out and the check may still pass; fortunately, more robust solutions exists in the study of *error correcting codes*.

Casting out 9s. Observe that a number is congruent to the sum of its digits modulo 9. (Can you show this? Hint: start by showing $10^n \equiv 1 \pmod 9$ for any $n \in \mathbb{N}^+$.) The same fact also holds modulo 3. This allows us to check if the computation is correct by quickly performing the same computation modulo 9. (Note that incorrect computations might still pass, so this check only increase our confidence that the computation is correct.)

Pseudorandom sequences. Randomized algorithms require a source of random numbers; where do they come from in a computer? Computers today take a small random "seed" (this can be the current time, or taken from the swap space), and expand it to a long string that "looks random". A standard technique is the linear congruential generator (LCG):

- Choose a modulus $m \in N^+$,
- a multiplier $a \in 2, 3, \ldots, m-1$, and
- an increment $c \in \mathbb{Z}_m = \{0, 1, \ldots, m-1\}$

Given a seed $x_0 \in \mathbb{Z}_m$, the LCG outputs a "random looking" sequence inductively defined by

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

The LCG is good enough (i.e., random enough) for some randomized algorithms. Cryptographic algorithms, however, have a much more stringent requirement for "random looking"; it must be the case that any adversarial party, any hacker on the internet, cannot tell apart a "pseudorandom" string from a truly random string. Can you see why the LCG is not a good pseudorandom generator? (Hint: the LCG follows a very specific pattern)

Encryption Encryption solves the classical cryptographic problem of *secure communication*. Suppose that Alice wants to send a private message to Bob; however, the channel between Alice and Bob is insecure, in the sense that there is an adversary Eve who listens in on everything sent between Alice and Bob

(later in the course we will discuss even more malicious behaviors than just listening in). To solves this problem, Alice and Bob agrees on a "secret code" (an encryption scheme) so that Alice may "scramble" her messages to Bob (an encryption algorithm) in a way that no one except Bob may "unscramble" it (a decryption algorithm).

Definition 3.19 (Private-Key Encryption Scheme)**.** A triplet of algorithms $(\text{Gen}, \text{Enc}, \text{Dec})$, a message space $\mathcal{M}$, and a key space $\mathcal{K}$, together is called a private-key encryption scheme if:

1. The key-generation algorithm, Gen is a randomized algorithm that returns a key, $k \leftarrow \text{Gen}$, such that $k \in \mathcal{K}$.
2. The encryption algorithm, $\text{Enc} : \mathcal{K} \times \mathcal{M} \to \{0,1\}^*$ is an algorithm that takes as input a key $k \in \mathcal{K}$ and a *plain-text* $m \in \mathcal{M}$ (the message), and outputs a *cipher-text* $c = \text{Enc}_k(m) \in \{0,1\}^*$.
3. The decryption algorithm, $\text{Dec} : \mathcal{K} \times \{0,1\}^* \to \mathcal{M}$ is an algorithm that takes as input a key $k \in \mathcal{K}$ and a cipher-text $c \in \{0,1\}^*$, and output a plain-text $m \in \mathcal{M}$.
4. The scheme is *correct*; that is, decrypting a valid cipher-text should output the original plain text. Formally we require that for all $m \in \mathcal{M}$, $k \in \mathcal{K}$, $\text{Dec}_k(\text{Enc}_k(m)) = m$.

To use a private encryption scheme, Alice and Bob first meet in advance and run $k \leftarrow \text{Gen}$ together to agree on the *secret key* k. The next time Alice has a private message m for Bob, she sends $c = \text{Enc}_k(m)$ over the insecure channel. Once Bob receives the cipher-text c, he decrypts it by running $m = \text{Dec}_k(c)$ to read the original message.

Example 3.20 (Caesar Cipher)**.** The Caesar Cipher is a private-key encryption scheme used by Julius Caesar to communicate with his generals; encryption is achieved by "shifting" each alphabet by some fixed amount (the key). Here is the formal description of the scheme. Let $\mathcal{M} = \{A, \ldots, Z\}^*$ and $\mathcal{K} = \{0, \ldots, 25\}$:

- Gen outputs a uniformly random key k from $\mathcal{K} = \{0, \ldots, 25\}$.
- Encryption shifts the alphabet of each letter in the plain-text by k:

$$\text{Enc}_k(m_1 m_2 \cdots m_n) = c_1 c_2 \cdots c_n, \text{where } c_i = m_i + k \bmod 26$$

- Decryption shifts each letter back:

$$\text{Dec}_k(c_1 c_2 \cdots c_n) = m_1 m_2 \cdots c_n, \text{where } m_i = c_i - k \bmod 26$$

For example, if $k = 3$, then we substitute each letter in the plain-text according to the following table:

plain-text: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
cipher-text: `DEFGHIJKLMNOPQRSTUVWXYZABC`

The message `GOODMORNING` encrypts to `JRRGPRUQLQJ`.

Claim 3.21. *The Caesar Cipher is a private-key encryption scheme.*

Proof. Correctness is trivial, since for all alphabets m and all keys k,

$$m = ((m + k) \bmod 26 - k) \bmod 26$$ ■

Nowadays, we know the Caesar Cipher is not a very good encryption scheme. There are numerous freely available programs or applets on-line that can crack the Caesar Cipher. (In fact, you can do it too! After all, there are only 26 keys to try.) The next example is on the other extreme; it is a *perfectly* secure private-key encryption scheme. We wait until a later chapter to formalize the notion of perfect secrecy; for now, we simply point out that at least the key length of the one-time pad grows with the message length (i.e., there is not just 26 keys).

Example 3.22 (One-Time Pad)**.** In the one-time pad encryption scheme, the key is required to be as long as the message. During encryption, the entire key is used to mask the plain-text, and therefore "perfectly hides" the plain-text. Formally, let $\mathcal{M} = \{0,1\}^n$, $\mathcal{K} = \{0,1\}^n$, and

- Gen samples a key $k = k_1 k_2 \cdots k_n$ uniformly from $\mathcal{K} = \{0,1\}^n$.
- $\text{Enc}_k(m_1 m_2 \cdots m_n) = c_1 c_2 \cdots c_n$, where $c_i = m_i + k_i \bmod 2$ (equivalently $c_i = m_i \oplus k_i$ where $\oplus$ denotes XOR).
- $\text{Dec}_k(c_1 c_2 \cdots c_n) = m_1 m_2 \cdots m_n$, where $m_i = c_i - k_i \bmod 2$ (again, it is equivalent to say $m_i = c_i \oplus k_i$).

To encrypt the message $m =$ `0100000100101011` under key $k =$ `1010101001010101`, simply compute

plain-text:	`0100000100101011`
$\oplus$ key:	`1010101001010101`
cipher-text:	`1110101101111110`

Claim 3.23. *The one-time pad is a private-key encryption scheme.*

Proof. Again correctness is trivial, since for $m_i \in \{0,1\}$ and all $k_i \in \{0,1\}$, $m_i = ((m_i + k_i) \bmod 2 - k_i) \bmod 2$ (equivalently, $m_i = m_i \oplus k_i \oplus k_i$). ■

Private-key encryption is limited by the precondition that Alice and Bob must meet in advance to (securely) exchange a private key. Is this an inherent cost for achieving secure communication?

First let us ask: can parties communicate securely without having secrets? Unfortunately, the answer is *impossible*. Alice must encrypt her message based on some secret key known only to Bob; otherwise, everyone can run the same decryption procedure as Bob to view the private message. Does this mean Alice has to meet with Bob in advance?

Fortunately, the answer this time around is no. The crux observation is that maybe we don't need the whole key to encrypt a message. Public-key cryptography, first proposed by Diffie and Hellman in 1976, splits the key into two parts: an encryption key, called **public-key**, and a decryption key, called the *secret-key*. In our example, this allows Bob to generate his own public and private key-pair without meeting with Alice. Bob can then publish his public-key for anyone to find, including Alice, while keeping his secret-key to himself. Now when Alice has a private message for Bob, she can encrypt it using Bob's public-key, and be safely assured that only Bob can decipher her message.

To learn more about public-key encryption, we need more number theory; in particular, we need to notion of *prime* numbers.

3.3 Primes

Primes are numbers that have the absolute minimum number of divisors; they are only divisible by themselves and 1. Composite numbers are just numbers that are not prime. Formally:

Definition 3.24 (Primes and Composites)**.** Let $p \in \mathbb{N}$ and $p \geq 2$. p is *prime* if its only positive divisors are 1 and p. Otherwise p is composite (i.e., there exists some a such that $1 < a < n$ and $a|n$).

Note that the definition of primes and composites exclude the numbers 0 and 1. Also note that, if n is composite, we may assume that there exists some a such that $1 < a \leq \sqrt{n}$ and $a|n$. This is because given a divisor $d|n$ with $1 < d < n$, then $1 < n/d < n$ is also a divisor of n; moreover, one of d or n/d must be at most $\sqrt{n}$.

Example 3.25. The first few primes are: 2,3,5,7,11,13,17,19.
The first few composites are: 4,6,8,9,10,12,14,15.

How can we determine if a number n is prime or not? This is called a primality test. Given the above observation, we can try to divide n by every positive integer $\leq \sqrt{n}$; this is not very efficient, considering that in today's cryptographic applications, we use 1024 or 2048-bit primes. A deterministic polynomial-time algorithm to for primality tests was not known until Agarwal, Saxena and Kayal constructed the first such algorithm in 2002; even after several improvements, the algorithm is still not fast enough to practically feasible. Fortunately, there is a much more efficient probabilistic algorithm for primality test; we will discuss this more later.

Distribution of Primes

How many primes are there? Euclid first showed that there are infinitely many primes.

Theorem 3.26 (Euclid)**.** *There are infinitely many primes.*

Proof. Assume the contrary that there exists a finite number of primes $p_1, \ldots, p_n$. Consider $q = p_1 p_2 \cdots p_n + 1$. By assumption, q is not prime. Let $a > 1$ be the smallest number that divides q. Then a must be prime (or else it could not be the smallest, by transitivity of divisibility), i.e., $a = p_i$ for some i (since $p_1, \ldots, p_n$ are all the primes).

We have $p_i | q$. Since $q = p_1 p_2 \cdots p_n + 1$ and p_i clearly divides $p_1 p_2 \cdots p_n$, we conclude by Corollary 3.4 that $p_i | 1$, a contradiction. ■

Not only are there infinitely many primes, primes are actually common (enough).

Theorem 3.27 (Prime Number Theorem)**.** *Let $\pi(N)$ be the number of primes $\leq N$. Then*

$$\lim_{N \to \infty} \frac{\pi(N)}{N/\ln N} = 1$$

We omit the proof, as it is out of the scope of this course. We can interpret the theorem as follows: there exists (small) constants c_1 and c_2 such that

$$c_1 N/logN \leq \pi(N) \leq c_2 N/logN$$

If we consider n-digit numbers, i.e., $0 \leq x < 10^n$, roughly $10^n / \log 10^n = 10^n / n$ numbers are prime. In other words, roughly $1/n$ fraction of n-digit numbers are prime.

Given that prime numbers are dense (enough), here is a method for finding a random n-digit prime:

- Pick a random (odd) n-digit number, x.
- Efficiently check if x is prime (we discuss how later).
- If x is prime, output x.
- Otherwise restart the procedure.

Roughly order n restarts would suffice.

Relative Primality

Primes are numbers that lack divisors. A related notion is relative primality, where a pair of number lacks *common* divisors.

Definition 3.28 (Relative Primality)**.** Two positive integers $a, b \in \mathbb{N}^+$ are relatively prime if $\gcd(a, b) = 1$.

Clearly, a (standalone) prime is relatively prime to any other number except a multiple of itself. From Theorem 3.13 (i.e., from Euclid's algorithm), we have an alternative characterization of relatively prime numbers:

Corollary 3.29. *Two positive integers $a, b \in \mathbb{N}^+$ are relatively prime if and only if there exists $s, t \in \mathbb{Z}$ such that $sa + tb = 1$.*

Corollary 3.29 has important applications in modular arithmetic; it guarantees the existence of certain multiplicative inverses (so that we may talk of modular division).

Theorem 3.30. *Let $a, b \in \mathbb{N}^+$. There exists an element a^{-1} such that $a^{-1} \cdot a \equiv 1 \pmod{b}$ if and only if a and b are relatively prime.*

a^{-1} is called the inverse of a modulo b; whenever such an element exists, we can "divide by a" modulo b by multiplying by a^{-1}. For example, 3 is the inverse of 7 modulo 10, because $7 \cdot 3 \equiv 21 \equiv 1 \pmod{10}$. On the other hand, 5 does not have an inverse modulo 10 (without relying on Theorem 3.30, we can establish this fact by simply computing $5 \cdot 1, 5 \cdot 2, \ldots, 5 \cdot 9$ modulo 10).

Proof of Theorem 3.30. **If direction.** If a and b are relatively prime, then there exists s, t such that $sa + tb = 1$ (Corollary 3.29). Rearranging terms,

$$sa = 1 - tb \equiv 1 \pmod{b}$$

therefore $s = a^{-1}$.

Only if direction. Assume there exists an element s such that $sa \equiv 1 \pmod{b}$. By definition this means there exists t such that $sa - 1 = tb$. Rearranging terms we have $sa + (-t)b = 1$, which implies $\gcd(a, b) = 1$. ■

Relative primality also has consequences with regards to divisibility.

Lemma 3.31. *If a and b are relatively prime and $a|bc$, then $a|c$.*

Proof. Because a and b are relatively prime, there exists s and t such that $sa + tb = 1$. Multiplying both sides by c gives $sac + tbc = c$. Since a divides the left hand side (a divides a and bc), a must also divides the right hand side (i.e., c). ■

The Fundamental Theorem of Arithmetic

The fundamental theorem of arithmetic states that we can factor any positive integer uniquely as a product of primes. We start with a lemma before proving the fundamental theorem.

Lemma 3.32. *If p is prime and $p \mid \prod_{i=1}^{n} a_i$, then there exist some $1 \leq j \leq n$ such that $p|a_j$.*

Proof. We proceed by induction. Let $P(n)$ be the statement: For every prime p, and every sequence $a_1, \ldots a_n$, if $p \mid \prod_{i=1}^{n} a_i$, then there exist some $1 \leq j \leq n$ such that $p|a_j$.

Base case: $P(1)$ is trivial ($j = 1$).

Inductive step: Assuming $P(n)$, we wish to show $P(n+1)$. Consider some prime p and sequence $a_1, .., a_{n+1}$ s.t. $p| \prod_{i=1}^{n+1} a_i$. We split into two cases, either $p \mid a_{n+1}$ or $\gcd(p, a_{n+1}) = 1$ (if the gcd was more than 1 but less than p, then p would not be a prime).

In the case that $p \mid a_{n+1}$ we are immediately done ($j = n + 1$). Otherwise, by Lemma 3.31, $p \mid \prod_{i=1}^{n} a_i$. We can then use the induction hypothesis to show that there exists $1 \leq j \leq n$ such that $p|a_j$. ■

Theorem 3.33 (Fundamental Theorem of Arithmetic)**.** *Every natural number $n > 1$ can be uniquely factored into a product of a sequence of non-decreasing primes, i.e., the* unique prime factorization*.*

For example, $300 = 2 \times 2 \times 3 \times 5 \times 5$.

Proof. We proceed by induction. Let $P(n)$ be the statement "n has a unique prime factorization".

Base case: $P(2)$ is trivial, with the unique factorization $2 = 2$.

Inductive Step: Assume $P(k)$ holds for all $2 \leq k \leq n - 1$. We will show $P(n)$ (for $n \geq 3$). If n is prime, then we have the factorization of $n = n$, and this is unique (anything else would contradict that n is prime). If n is composite, we show existence and uniqueness of the factorization separately:

Existence. If n is composite, then there exists a, b such that $2 \leq a, b \leq n - 1$ and $n = ab$. Apply the induction hypothesis $P(a)$ and $P(b)$ to get their respective factorization, and "merge them" for a factorization of n.

Uniqueness. Suppose the contrary that n has two different factorizations,

$$n = \prod_{i=1}^{n} p_i = \prod_{j=1}^{m} q_j$$

where $n, m \geq 2$ (otherwise n would be prime). Because $p_1|n = \prod_j q_j$, by Lemma 3.32 there is some j_0 such that $p_1|q_{j_0}$. Since p_1 and q_{j_0} are both primes, we have $p_1 = q_{j_0}$. Consider the number $n' = n/p_1$ which can be factorized as

$$n' = \prod_{i=2}^{n} p_i = \prod_{j=1, j\neq j_0}^{m} q_j$$

Since $1 < n' = n/p_1 < n$, the induction hypothesis shows that the two factorizations of n' are actually the same, and so the two factorization of n are also the same (adding back the terms $p_1 = q_{j_0}$). ■

Open Problems

Number theory is a field of study that is rife with (very hard) open problems. Here is a small sample of open problems regarding primes.

Goldbach's Conjecture, first formulated way back in the 1700's, states that any positive even integer other than 2 can be expressed as the sum of two primes. For example $4 = 2 + 2$, $6 = 3 + 3$, $8 = 3 + 5$ and $22 = 5 + 17$. With modern computing power, the conjecture has been verified for all even integers up to $\approx 10^{17}$.

The **Twin Prime Conjecture** states that there are infinitely pairs of primes that differ by 2 (called twins). For example, 3 and 5, 5 and 7, 11 and 13 or 41 and 43 are all twin primes. A similar conjecture states that there are infinitely many safe primes or Sophie-Germain primes — pairs of primes of the form p and $2p+1$ (p is called the Sophie-Germain prime, and $2p+1$ is called the safe prime). For example, consider 3 and 7, 11 and 23, or 23 and 47. In cryptographic applications, the use of safe primes sometimes provide more security guarantees.

3.4 The Euler ϕ Function

Definition 3.34 (The Euler ϕ Function)**.** Given a positive integer $n \in \mathbb{N}^+$, define $\phi(n)$ to be the number of integers $x \in \mathbb{N}^+$, $x \leq n$ such that $\gcd(x, n) = 1$, i.e., the number of integers that are relatively prime with n.

For example, $\phi(6) = 2$ (the relatively prime numbers are 1 and 5), and $\phi(7) = 6$ (the relatively prime numbers are 1, 2, 3, 4, 5, and 6). By definition $\phi(1) = 1$ (although this is rather uninteresting). The Euler ϕ function can be computed easily on any integer for which we know the unique prime factorization (computing the unique prime factorization itself may be difficult). In fact, if the prime factorization of n is $n = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$, then

$$\phi(n) = n \prod_i \left(1 - \frac{1}{p_i}\right) = \prod_i (p_i^{k_i} - p_i^{k_i - 1}) \tag{3.5}$$

While we won't prove (3.5) here (it is an interesting counting exercise), we do state and show the following special cases.

Claim 3.35. *If p is a prime, then $\phi(p) = p - 1$. If $n = pq$ where $p \neq q$ are both primes, then $\phi(n) = (p-1)(q-1)$.*

Proof. If p is prime, then the numbers $1, 2, \ldots, p-1$ are all relatively prime to p. Therefore $\phi(p) = p - 1$.

If $n = pq$ with $p \neq q$ both prime, then among the numbers $1, 2, \ldots, n = pq$, there are exactly q multiples of p (they are $p, 2p, \ldots, n = qp$). Similarly, there are exactly p multiples of q. Observe that that other than multiples of p or q, the rest of the numbers are relatively prime to n; also observe that we have counted $n = pq$ twice. Therefore

$$\phi(n) = n - p - q + 1 = pq - p - q + 1 = (p-1)(q-1) \qquad \blacksquare$$

The Euler ϕ function is especially useful in modular exponentiation, due to Euler's Theorem:

Theorem 3.36. *Given $a, n \in \mathbb{N}^+$, if $\gcd(a, n) = 1$, then*

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Proof. Let $X = \{x \mid x \in N^+, x \leq n, \gcd(x, n) = 1\}$. By definition $|X| = \phi(n)$. Let $aX = \{ax \bmod n \mid x \in X\}$. For example, if $n = 10$, then $X = \{1, 3, 7, 9\}$, and

$$aX = \{3 \bmod 10, 9 \bmod 10, 21 \bmod 10, 27 \bmod 10\} = \{3, 9, 1, 7\}$$

By Theorem 3.30, X is the set of all numbers that have multiplicative inverses modulo n; this is useful in the rest of the proof.

We first claim that $X = aX$ (this does indeed hold in the example). We prove this by showing $X \subseteq aX$ and $aX \subseteq X$.

$X \subseteq aX$. Given $x \in X$, we will show that $x \in aX$. Consider the number $a^{-1}x \bmod n$ (recall that a^{-1} is the multiplicative inverse of a, and exists since $\gcd(a, n) = 1$). We claim that $a^{-1}x \bmod n \in X$, since it has the multiplicative inverse $x^{-1}a$. Consequently, $a(a^{-1}x) \equiv x \pmod n \in aX$.

$aX \subseteq X$. We can give a similar argument as above[1]. Given $y \in aX$, we will show that $y \in X$. This can be done by constructing the multiplicative inverse of y. We know y is of the form ax for some $x \in X$, so the multiplicative inverse of y is $x^{-1}a^{-1}$.

Knowing $aX = X$, we have:

$$\prod_{x \in X} x \equiv \prod_{y \in aX} y \equiv \prod_{x \in X} ax \pmod n$$

Since each $x \in X$ has an inverse, we can multiply both sides with all those inverses (i.e., divide by $\prod_{x \in X} s$):

$$1 \equiv \prod_{x \in X} a \pmod n$$

Since $|X| = \phi$, we have just shown that $a^{\phi(n)} \equiv 1 \pmod n$. ■

We give two corollaries.

Corollary 3.37 (Fermat's Little Theorem)**.** *If p is a prime and* $\gcd(a, p) = 1$*, then* $a^{p-1} \equiv 1 \pmod p$.

Proof. This directly follows from the theorem and $\phi(p) = p - 1$. ■

Corollary 3.38. *If* $\gcd(a, n) = 1$*, then*

$$a^x \equiv a^{x \bmod \phi(n)} \pmod n$$

[1] In class we showed this differently. Observe that $|aX| \leq |X|$ (since elements in aX are "spawned" from X). Knowing that $X \subseteq aX$, $|aX| \leq |X|$, and the fact that these are finite sets allows us to conclude that $aX \subseteq X$ (in fact we may directly conclude that $aX = X$).

Proof. Let $x = q\phi(n) + r$ be the result of dividing x by $\phi(n)$ with remainder (recall that r is exactly $x \bmod \phi(n)$). Then

$$a^x = a^{q\phi(N)+r} = (a^{\phi(N)})^q \cdot a^r \equiv 1^q \cdot a^r = a^{x \bmod \phi(n)} \pmod{n}$$ ■

Example 3.39. Euler's function can speed up modular exponentiation by a lot. Let $n = 21 = 3 \times 7$. We have $\phi(n) = 2 \times 6 = 12$. Then

$$2^{999} \equiv 2^{999 \bmod 12} = 2^3 = 8 \pmod{21}$$

Application to Probabilistic Primality Checking

We have mentioned on several occasions that there is an efficient probabilistic algorithm for checking primality. The algorithm, on input a number n, has the following properties:

- If n is prime, then algorithm always outputs YES
- If n is not prime, then with some probability, say $1/2$, the algorithm may still output YES incorrectly.

Looking at it from another point of view, if the algorithm ever says n is not prime, then n is definitely not prime. With such an algorithm, we can ensure that n is prime with very high probability: run the algorithm 200 times, and believe n is prime if the output is always YES. If n is prime, we always correctly conclude that it is indeed prime. If n is composite, then we would only incorrectly view it as a prime with probability $(1/2)^{200}$ (which is so small that is more likely to encounter some sort of hardware error).

How might we design such an algorithm? A first approach, on input n, is to pick a random number $1 < a < n$ and output YES if and only if $\gcd(a, n) = 1$. Certainly if n is prime, the algorithm will always output YES. But if n is not prime, this algorithm may output YES with much too high probability; in fact, it outputs YES with probability $\approx \phi(n)/n$ (this can be too large if say $n = pq$, and $\phi(n) = (p-1)(q-1)$).

We can design a similar test relying on Euler's Theorem. On input n, pick a random $1 < a < n$ and output YES if and only if $a^{n-1} \equiv 1 \pmod{n}$. Again, if n is prime, this test will always output YES. What if n is composite? For most composite numbers, the test does indeed output YES with sufficiently small probability. However there are some composites, called Carmichael numbers or pseudo-primes, on which this test always outputs YES incorrectly (i.e., a Carmichael number n has the property that for all $1 < a < n$, $a^{n-1} \equiv 1 \pmod{n}$, yet n is not prime).

By adding a few tweaks to the above algorithm, we would arrive at the Miller-Rabin primality test that performs well on all numbers (this is out of the scope of this course). For now let us focus on computing a^{n-1}. The naïve way of computing a^{n-1} requires $n-1$ multiplications — in that case we might as well just divide n by all numbers less than n. A more clever algorithm is to do *repeated squaring*:

Algorithm 2 ExpMod(x, e, n), computing $x^e \bmod n$

```
if e = 0 then
  return 1;
else
  return ExpMod(x, e div 2, n)^2 · x^(e mod 2) mod n;
end if
```

The correctness of ExpMod is based on the fact the exponent e can be expressed as $(e \text{ div } 2) \cdot 2 + e \bmod 2$ by the division algorithm, and therefore

$$x^e = x^{(e \text{ div } 2)\cdot 2 + e \bmod 2} = \left(x^{e \text{ div } 2}\right)^2 \cdot x^{e \bmod 2}$$

To analyse the efficiency of ExpMod, observe that $x^{e \bmod 2}$ is easy to compute (it is either $x^1 = x$ or $x^0 = 1$), and that the recursion has depth $\approx \log_2 e$ since the exponent e is halved in each recursive call. The intuition behind ExpMod is simple. By repeated squaring, it is must faster to compute exponents that are powers of two, e.g., to compute x^{16} requires squaring four times: $x \to x^2 \to x^4 \to x^8 \to x^{16}$. Exponents that are not powers of two can first be split into sums of powers of two; this is the same concept as binary representations for those who are familiar with it. As an example, suppose we want to compute $4^{19} \bmod 13$. First observe that $19 = 16 + 2 + 1$ (the binary representation of 19 would be 1011_2). By repeated squaring, we first compute:

$$4^2 \bmod 13 = 16 \bmod 13 = 3 \qquad 4^4 \bmod 13 = 3^2 \bmod 13 = 9$$
$$4^8 \bmod 13 = 9^2 \bmod 13 = 3 \qquad 4^{16} \bmod 13 = 3^2 \bmod 13 = 9$$

Now we can compute

$$\begin{aligned} 4^{19} \bmod 13 = 4^{16+2+1} \bmod 13 &= 4^{16} \cdot 4^2 \cdot 4^1 \bmod 13 \\ &= 9 \cdot 3 \cdot 4 \bmod 13 = 4 \end{aligned}$$

The takeaway of this section is that primality testing can be done efficiently, in time polynomial in the length (number of digits) of the input number n (i.e., in time polynomial in $\log n$).

3.5 Public-Key Cryptosystems and RSA

In this section we formally define public-key cryptosystems. We describe the RSA cryptosystem as an example, which relies on many number theoretical results in this chapter.

Recall our earlier informal discussion on encryption: Alice would like to send Bob a message over a public channel where Eve is eavesdropping. A private-key encryption scheme would require Alice and Bob to meet in advance to jointly generate and agree on a secret key. On the other hand a public-key encryption scheme, first proposed by Diffie and Hellman, has two keys: a public-key for

encrypting, and a private-key for decrypting. Bob can now generate both keys by himself, and leave the public-key out in the open for Alice to find; when Alice encrypts her message using Bob's public-key, only Bob can decrypt and read the secret message.

Definition 3.40. A triplet of algorithms $(\text{Gen}, \text{Enc}, \text{Dec})$, a key space $\mathcal{K} \subseteq \mathcal{K}_{\mathbf{pk}} \times \mathcal{K}_{\mathbf{sk}}$ (each key is actually a pair of public and secret keys), and a sequence of message spaces indexed by public-keys $\mathcal{M} = \{\mathcal{M}_{pk}\}_{pk \in \mathcal{K}_{\mathbf{pk}}}$, together is called a public-key encryption scheme if:

1. The key-generation algorithm, Gen is a randomized algorithm that returns a pair of keys, $(pk, sk) \leftarrow \text{Gen}$, such that $(pk, sk) \in \mathcal{K}$ (and so $pk \in \mathcal{K}_{\mathbf{pk}}, sk \in \mathcal{K}_{\mathbf{sk}}$).

2. The encryption algorithm takes as input a public-key $pk \in \mathcal{K}_{\mathbf{pk}}$ and a *plain-text* $m \in \mathcal{M}_{pk}$ (the message), and outputs a *cipher-text* $c = \text{Enc}_{pk}(m) \in \{0,1\}^*$.

3. The decryption algorithm takes as input a secret-key $k \in \mathcal{K}_{\mathbf{sk}}$ and a cipher-text $c \in \{0,1\}^*$, and output a plain-text $m = \text{Dec}_{sk}(c)$.

4. The scheme is *correct*; that is, decrypting a valid cipher-text with the correct pair of keys should output the original plain text. Formally we require that for all $(pk, sk) \in \mathcal{K}$, $m \in \mathcal{M}_{pk}$, $\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$.

The RSA Cryptosystem

The RSA cryptosystem, conceived by Rivest, Shamir and Adleman, uses modular exponentiation to instantiate a public-key crypto-system. Given a security parameter n, the plain RSA public-key encryption scheme is as follows:

- $\text{Gen}(n)$ picks two random n-bit primes p and q, set $N = pq$, and pick a random e such that $1 < e < n$, $\gcd(e, \phi(N)) = 1$. The public key is $pk = (N, e)$, while the secret key is $sk = (p, q)$ (the factorization of N). N is called the modulus of the scheme.

- Through the definition of Gen, we already have a implicit definition of the key-space $\mathcal{K}$. The message space for a public-key $pk = (N, e)$ is: $\mathcal{M}_{pk} = \{m \mid 0 < m < N, \gcd(m, N) = 1\}$.

- Encryption and decryption are defined as follows:

$$\begin{aligned} \text{Enc}_{pk}(m) &= m^e \bmod N \\ \text{Dec}_{sk}(c) &= c^d \bmod N, \text{where } d = e^{-1} \bmod \phi(N) \end{aligned}$$

Correctness of RSA

First and foremost we should verify that all three algorithms, Gen, Enc and Dec, can be efficiently computed. Gen involves picking two n-bit primes, p and q, and an exponent e relatively prime to pq; we covered generating random primes in Section 3.3, and choosing e is simple: just make sure e is not a multiple of p or q (and a random e would work with very high probability). Enc and Dec are both modular exponentiations; we covered that in Section 3.4. Dec additionally requires us to compute $d = e^{-1} \bmod \phi(N)$; knowing the secret-key, which contains the factorization of N, it is easy to compute $\phi(N) = (p-1)(q-1)$, and then compute d using the extended GCD algorithm and Theorem 3.30.

Next, let us verify that decryption is able to recover encrypted messages. Given a message m satisfying $0 < m < N$ and $\gcd(m, N) = 1$, we have:

$$\begin{aligned}
\mathrm{Dec}(\mathrm{Enc}(m)) &= ((m^e) \bmod N)^d \bmod N \\
&= m^{ed} \bmod N \\
&= m^{ed \bmod \phi(N)} \bmod N && \text{by Corollary 3.38} \\
&= m^1 \bmod N && \text{since } d = e^{-1} \bmod \phi(N) \\
&= m \bmod N = m
\end{aligned}$$

This calculation also shows why the message space is restricted to $\{m \mid 0 < m < N, \gcd(m, N) = 1\}$: A message m must satisfy $\gcd(m, N) = 1$ so that we can apply Euler's Theorem, and m must be in the range $0 < m < N$ so that when we recover $m \bmod N$, it is actually equal to the original message m.

Security of RSA

Let us informally discuss the security of RSA encryption. What stops Eve from decrypting Alice's messages? The assumption we make is that without knowing the secret key, it is hard for Eve to compute $d = e^{-1} \bmod \phi(N)$. In particular, we need to assume the *factoring conjecture*: there is no efficient algorithm that factor numbers N that are products of two equal length primes p and q (formally, efficient algorithm means any probabilistic algorithm that runs in time polynomial in the length of N, i.e., the number of digits of N). Otherwise Eve would be able to recover the secret-key and decrypt in the same way as Bob would.

There is another glaring security hole in the our description of the RSA scheme: the encryption function is deterministic. What this means is that once the public-key is fixed, the encryption of each message is unique! For example, there is only one encryption for the word "YES", and one encryption for the word "NO", and anyone (including Eve) can compute these encryptions (it is a public-key scheme after all). If Alice ever sends an encrypted YES or NO answer to Bob, Eve can now completely compromise the message.

One solution to this problem is for Alice to pad each of her message m with a (fairly long) random string; she than encrypts the resulting padded message m'

as before, outputting the cipher-text $(m')^e \bmod N$ (now the whole encryption procedure is randomized). This type of "padded RSA" is implemented in practice.

On generating secret keys. Another security concern (with any key-based scheme) is the quality of randomness used to generate the secret-keys. Let us briefly revisit the Linear Congruential Generator, a pseudorandom generator that we remarked should not be used for cryptographic applications. The LCG generates a sequence of numbers using the recurrence:

$$x_0 = \text{random seed}$$
$$x_i = ax + c \bmod M$$

In C++, we have $a = 22695477$, $c = 1$, and $M = 2^{32}$. Never mind the fact that the sequence $(x_0, x_1, x_2, \dots)$ has a pattern (that is not very random at all). Because there are only 2^{32} starting values for x_0, we can simply try them all, and obtain the secret-key of any RSA key-pair generated using LCG and C++.[2]

Padded RSA

We have already discussed why a padded scheme for RSA is necessary for security. A padded scheme also has another useful feature; it allows us to define a message space that does not depend on the choice of the public-key (e.g., it would be tragic if Alice could not express her love for Bob simply because Bob chose the wrong key). In real world implementations, designing the padding scheme is an engineering problem with many practical considerations; here we give a sample scheme just to illustrate how padding can be done. Given a security parameter n, a padded RSA public-key encryption scheme can proceed as follows:

- $\text{Gen}(n)$ picks two random n-bit primes, $p, q > 2^{n-1}$, and set $N = pq$, and pick a random e such that $1 < e < n$, $\gcd(e, \phi(N)) = 1$. The public key is $pk = (N, e)$, while the secret key is $sk = (p, q)$ (the factorization of N). N is called the modulus of the scheme.
- Through the definition of Gen, we already have a implicit definition of the key-space $\mathcal{K}$. The message space is simply $\mathcal{M} = \{m \mid 0 \leq m < 2^n\} = \{0,1\}^n$, the set of n-bit strings.
- Encryption is probabilistic. Given public-key $pk = (N, e)$ and a message $m \in \mathcal{M}$, pick a random $(n-2)$-bit string r and let $r\|m$ be the concatenation of r and m, interpreted as an integer $0 \leq r\|m < 2^{2n-2} < N$. Furthermore, if $r\|m = 0$ or if $\gcd(r\|m, N) = 1$, we re-sample the random string r until it is not so. The output cipher-text is then

$$\text{Enc}_{pk}(m) = (r\|m)^e \bmod N$$

[2] In Java $M = 2^{48}$, so the situation improves a little bit.

- Decryption can still be deterministic. Given secret-key sk and cipher-text c, first decrypt as in plain RSA, i.e., $m' = c^d \bmod N$ where $d = e^{-1} \bmod \phi(N)$, and output the n right-most bits of m' as the plain-text.

RSA signatures

We end the section with another cryptographic application of RSA: digital signatures. Suppose Alice wants to send Bob a message expressing her love, "I love you, Bob", and Alice is so bold and confident that she is not afraid of eavesdroppers. However Eve is not just eavesdropping this time, but out to sabotage the relationship between Alice and Bob. She sees Alice's message, and changes it to "I hate you, Bob" before it reaches Bob. How can cryptography help with this sticky situation? A digital signature allows the sender of a message to "sign" it with a signature; when a receiver verifies the signature, he or she can be sure that the message came from the sender and has not been tampered.

In the RSA signature scheme, the signer generates keys similar to the RSA encryption scheme; as usual, the signer keeps the secret-key, and publishes the public-key. To sign a message m, the signer computes:

$$\sigma_m = m^d \bmod N$$

Anyone that receives a message m along with a signature σ can perform the following check using the public-key:

$$\sigma^e \bmod N \stackrel{?}{=} m$$

The correctness and basic security guarantees of the RSA signature scheme is the same as the RSA encryption scheme. Just as before though, there are a few security concerns with the scheme as described.

Consider this attack. By picking the signature σ first, and computing $m = \sigma^e$ anyone can forge a signature, although the message m is most likely meaningless (what if the attacker gets lucky?). Or suppose Eve collects two signatures, (m_1, σ_1) and (m_2, σ_2); now she can construct a new signature $(m = m_1 \cdot m_2 \bmod N, \sigma = \sigma_1 \cdot \sigma_2 \bmod N)$ (very possible that the new message m is meaningful). To prevents these two attacks, we modify the signature scheme to first transform the message using a "crazy" function H (i.e., $\sigma_m = H(m)^d \bmod N$).

Another important consideration is how do we sign large messages (e.g., lengthy documents)? Certainly we do not want to increase the size of N. If we apply the same solution as we did for encryption — break the message into chunks and sign each chunk individually — then we run into another security hole. Suppose Alice signed the sentences "I love you, Bob" and "I hate freezing rain" by signing the individual words; then Eve can collect and rearrange these signatures to produce a signed copy of "I hate you, Bob". The solution again relies on the crazy hash function H: we require H to accept arbitrary large messages as input, and still output a hash $< N$. A property that H must have

is *collision resistance*: it should be hard to find two messages, m_1 and m_2, that hash to the same thing $H(m_1) = H(m_2)$ (we wouldn't want "I love you, Bob" and "I hate you, Bob" to share the same signature).

Chapter 4

Counting

"How do I love thee? Let me count the ways."
– Elizabeth Browning

Counting is a basic mathematical tool that has uses in the most diverse circumstances. How much RAM can a 64-bit register address? How many poker hands form full houses compared to flushes? How many ways can ten coin tosses end up with four heads? To count, we can always take the time to enumerate all the possibilities; but even just enumerating all poker hands is already daunting, let alone all 64-bit addresses. This chapter covers several techniques that serve as useful short cuts for counting.

4.1 The Product and Sum Rules

The product and sum rules represent the most intuitive notions of counting. Suppose there are $n(A)$ ways to perform task A, and regardless of how task A is performed, there are $n(B)$ ways to perform task B. Then, there are $n(A) \cdot n(B)$ ways to perform both task A and task B; this is the **product rule**. This can generalize to multiple tasks, e.g., $n(A) \cdot n(B) \cdot n(C)$ ways to perform task A, B, and C, as long as the *independence* condition holds, e.g., the number of ways to perform task C does not depend on how task A and B are done.

Example 4.1. On an 8×8 chess board, how many ways can I place a pawn and a rook? First I can place the pawn anywhere on the board; there are 64 ways. Then I can place the rook anywhere except where the pawn is; there are 63 ways. In total, there are $64 \times 63 = 4032$ ways.

Example 4.2. On an 8×8 chess board, how many ways can I place a pawn and a rook so that the rook does not threaten the pawn? First I can place the rook anywhere on the board; there are 64 ways. At the point, the rook takes up on square, and threatens 14 others (7 in its row and 7 in its column). Therefore I can then place the pawn on any of the $64 - 14 - 1 = 49$ remaining squares. In total, there are $64 \times 49 = 3136$ ways.

Example 4.3. If a finite set S has n elements, then $|\mathcal{P}(S)| = 2^n$. We have seen a proof of this by induction; now we will see a proof using the product rule. $\mathcal{P}(S)$ is the set of all subsets of S. To form a subset of S, each of the n elements can either be in the subset or not (2 ways). Therefore there are 2^n possible ways to form unique subsets, and so $|\mathcal{P}(S)| = 2^n$.

Example 4.4. How many legal configurations are there in the towers of Hanoi? Each of the n rings can be on one of three poles, giving us 3^n configurations. Normally we would also need to count the height of a ring relative to other rings on the same pole, but in the case of the towers of Hanoi, the rings sharing the same pole must be ordered in a *unique* fashion: from small at the top to large at the bottom.

The **sum rule** is probably even more intuitive than the product rule. Suppose there are $n(A)$ ways to perform task A, and distinct from these, there are $n(B)$ ways to perform task B. Then, there are $n(A) + n(B)$ ways to perform task A or task B. This can generalize to multiple tasks, e.g., $n(A)+n(B)+n(C)$ ways to perform task A, B, or C, as long as the *distinct* condition holds, e.g., the ways to perform task C are different from the ways to perform task A or B.

Example 4.5. To fly from Ithaca to Miami you must fly through New York or Philadelphia. There are 5 such flights a day through New York, and 3 such flights a day through Philadelphia. How many different flights are there in a day that can take you from Ithaca to get to Miami? The answer is $5 + 3 = 8$.

Example 4.6. How many 4 to 6 digit pin codes are there? By the product rule, the number of distinct n digit pin codes is 10^n (each digit has 10 possibilities). By the sum rule, we have $10^4 + 10^5 + 10^6$ number of 4 to 6 digit pin codes (to state the obvious, we have implicitly used the fact that every 4 digit pin code is different from every 5 digit pin code).

4.2 Permutations and Combinations

Our next tools for counting are permutations and combinations. Given n distinct objects, how many ways are there to "choose" r of them? Well, it depends on whether the r chosen objects are *ordered* or not. For example, suppose we deal three cards out of a standard 52-card deck. If we are dealing one card each to Alice, Bob and Cathy, then the order of the cards being dealt matters; this is called a **permutation** of 3 cards. On the other hand, if we are dealing all three cards to Alice, then the order of the cards being dealt does not matter; this is called a **combination** or 3 cards.

Permutations

Definition 4.7. A permutation of a set A is an *ordered* arrangement of the elements in A. An ordered arrangement of just r elements from A is called

an r-permutation of A. For non-negative integers $r \leq n$, $P(n, r)$ denotes the number of r-permutations of a set with n elements.

What is $P(n, r)$? To form an r-permutation from a set A of n elements, we can start by choosing any element of A to be the first in our permutation; there are n possibilities. The next element in the permutation can be any element of A except the one that is already taken; there are $n-1$ possibilities. Continuing the argument, the final element of the permutation will have $n-(r-1)$ possibilities. Applying the product-rule, we have

Theorem 4.8.

$$P(n, r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}\ .^{1}$$

Example 4.9. How many one-to-one functions are there from a set A with m elements to a set B with n elements? If $m > n$ we know there are no such one-to-one functions. If $m \leq n$, then each one-to-one function f from A to B is a m-permutation of the elements of B: we choose m elements from B in an ordered manner (e.g., first chosen element is the value of f on the first element in A). Therefore there are $P(n, m)$ such functions.

Combinations

Let us turn to unordered selections.

Definition 4.10. An unordered arrangement of r elements from a set A is called an r-combination of A. For non-negative integers $r \leq n$, $C(n, r)$ or $\binom{n}{r}$ denotes the number of r-combinations of a set with n elements. $C(n, r)$ is also called the binomial coefficients (we will soon see why).

For example, how many ways are there to put two pawns on a 8×8 chess board? We can select 64 possible squares for the first pawn, and 63 possible remaining squares for the second pawn. But now we are over counting, e.g., choosing squares $(b5, c8)$ is the same as choosing $(c8, b5)$ since the two pawns are identical. Therefore we divide by 2 to get the correct count: $64 \times 63/2 = 2016$. More generally,

Theorem 4.11.

$$C(n, r) = \frac{n!}{(n-r)!r!}$$

Proof. Let us express relate $P(n, r)$ with $C(n, r)$. It must be that $P(n, r) = C(n, r)P(r, r)$, because to select an r-permutation from n elements, we can first

[1] Recall that $0! = 1$.

selected an *unordered* set of r elements, and then select an ordering of the r elements. Expanding the expression gives:

$$C(n,r) = \frac{P(n,r)}{P(r,r)} = \frac{n!/(n-r)!}{r!} = \frac{n!}{(n-r)!r!} \quad \blacksquare$$

Example 4.12. How many poker hands (i.e., sets of 5 cards) can be dealt from a standard deck of 52 cards? Exactly $C(52,5) = 52!/(47!5!)$.

Example 4.13. How many full houses (3 of a kind and 2 of another) can be dealt from a standard deck of 52 cards? Recall that we have 13 denominations (ace to king), and 4 suites (spades, hearts, diamonds and clubs). To count the number of full houses, we may

- First pick a denomination for the "3 of a kind": there are 13 choices.
- Pick 3 cards from this denomination (out of 4 suites): there are $C(4,3) = 4$ choices.
- Next pick a denomination for the "2 of a kind": there are 12 choices left (different from the "3 of a kind").
- Pick 2 cards from this denomination: there are $C(4,2) = 6$ choices.

So in total there are $13 * 4 * 12 * 6 = 3744$ possible full houses.

Balls and Urns

How many ways are there to put n balls into k urns? This classical counting problem has many variations. For our setting, we assume that the urns are distinguishable (e.g., numbered). If the balls are also distinguishable, then this is a simple application of the product rule: each ball can be placed into k possible urns, resulting in a total of k^n possible placements.

What if the balls are indistinguishable? Basically we need to assign a number to each urn, representing the numbers of balls in the urn, so that the sum of the numbers is n. Suppose we line up the n balls, and put $k-1$ delimiters between some of the adjacent balls. Then we would have exactly what we need: n balls split among k distinct urns (see Figure 4.1). The number of ways to place the delimiters is as simple as choosing $k-1$ delimiters among $n+k-1$ positions (total number of positions for both balls and delimiters), i.e., $C(n+k-1, k-1)$.

Example 4.14. How many solutions are there to the equation $x+y+z = 100$, if $x, y, z \in \mathbb{N}$? This is just like having 3 distinguishable urns (x, y and z) and 100 indistinguishable balls, so there are $C(102,2)$ solutions.

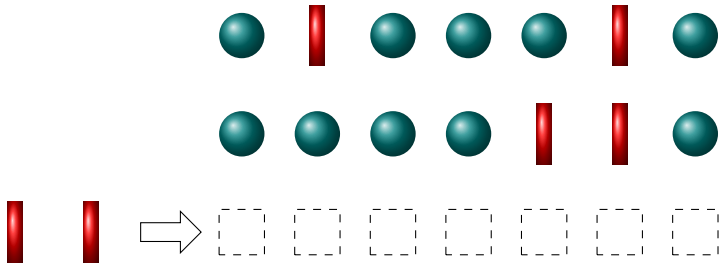


Figure 4.1: Suppose there are 5 balls and 3 urns. Using the delimiter idea, the first row represents the configuration $(1, 3, 1)$ (1 ball in the first urn, 3 balls in the second , and 1 ball in the third). The second row represents the configuration $(4, 0, 1)$ (4 balls in the first urn, none in the second, and 1 ball in the third). In general, we need to choose 2 positions out of 7 as delimiters (the rest of the positions are the 5 balls).

4.3 Combinatorial Identities

There are many identities involving combinations. These identities are fun to learn because they often represents different ways of counting the same thing; one can also prove these identities by churning out the algebra, but that is boring. We start with a few simple identities.

Lemma 4.15. *If $0 \leq k \leq n$, then $C(n, k) = C(n, n - k)$.*

Proof. Each unordered selection of k elements has a unique complement: an unordered selection of $n - k$ elements. So instead of counting the number of selections of k elements from n, we can count the number of selections of $n - k$ elements from n (e.g., to deal 5 cards from a 52 card deck is the same as to throw away $52 - 5 = 47$ cards).

An algebraic proof of the same fact (without much insight) goes as follows:

$$C(n, k) = \frac{n!}{(n-k)!k!} = \frac{n!}{(n-(n-k))!(n-k)!} = C(n, n-k) \qquad \blacksquare$$

Lemma 4.16 (Pascal's Identity)**.** *If $0 < k \leq n$, then $C(n + 1, k) = C(n, k - 1) + C(n, k)$.*

Proof. Here is another way to choose k elements from $n + 1$ total elements. Either the $n + 1^{\text{st}}$ element is chosen or not:

- If it is, then it remains to choose $k-1$ elements from the first n elements.
- If it isn't, then we need to choose all k elements from the first n elements.

By the sum rule, we have $C(n + 1, k) = C(n, k - 1) + C(n, k)$. $\blacksquare$

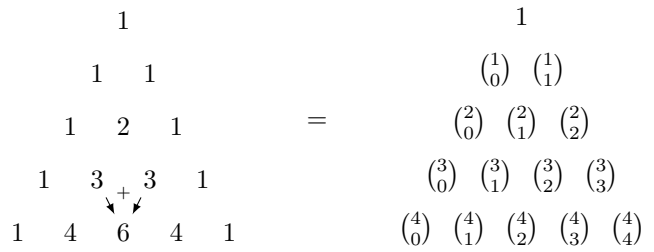


Figure 4.2: Pascal's triangle contains the binomial coefficients $C(n,k)$ ordered as shown in the figure. Each entry in the figure is the sum of the two entries on top of it (except the entries on the side which are always 1).

Pascals identity, along with the initial conditions $C(n,0) = C(n,n) = 1$, gives a recursive way of computing the binomial coefficients $C(n,k)$. The recursion table is often written as a triangle, called Pascal's Triangle; see Figure 4.2.

Here is another well-known identity.

Lemma 4.17. $\sum_{k=0}^{n} C(n,k) = 2^n$.

Proof. Let us once again count the number of possible subsets of a set of n elements. We have already seen by induction and by the product rule that there are 2^n such subsets; this is the RHS.

Another way to count is to use the sum rule:

$$\#\text{ of subsets} = \sum_{k=0}^{n} \#\text{ of subsets of size } k = \sum_{k=0}^{n} C(n,k)$$

This is the LHS. ∎

The next identity is more tricky:

Lemma 4.18. $\sum_{k=0}^{n} kC(n,k) = n2^{n-1}$.

Proof. The identity actually gives two ways to count the following problem: given n people, how many ways are there to pick a committee of any size, and then pick a chairperson of the committee? The first way to count is:

- Use the sum rule to count committees of different sizes individually.
- For committees of size k, there are $C(n,k)$ ways of choosing the committee, and independently, k ways of choosing a chairperson from the committee.

- This gives a total of $\sum_{k=0}^{n} kC(n,k)$ possibilities; this is the LHS.

The second way to count is:

- Pick the chairman first; there are n choices.
- For the remaining $n-1$ people, each person can either be part of the committee or not; there are 2^{n-1} possibilities.
- This gives a total of $n2^{n-1}$ possibilities; this is the RHS. ■

A similar trick can be used to prove Vandermonde's Identity.

Lemma 4.19 (Vandermondes Identity)**.** *If $r \leq m$ and $r \leq n$, then*

$$C(m+n,r) = \sum_{k=0}^{r} C(m,r-k)C(n,k)$$

Proof. Let M be a set with m elements and N be a set with n elements. Then the LHS represents the number of possible ways to pick r elements from M and N together. Equivalently, we can count the same process by splitting into r cases (the sum rule): let k range from 0 to r, and consider picking $r-k$ elements from M and k elements from M. ■

The next theorem explains the name "binomial coefficients": the combination function $C(n,k)$ are also the coefficient of powers of the simplest binomial, $(x+y)$.

Theorem 4.20 (The Binomial Theorem)**.** *For $n \in \mathbb{N}$,*

$$(x+y)^n = \sum_{k=0}^{n} C(n,k)x^{n-k}y^k$$

Proof. If we manually expand $(x+y)^n$, we would get 2^n terms with coefficient 1 (each term corresponds to choosing x or y from each of the n factors). If we then collect these terms, how many of them have the form $x^{n-k}y^k$? Terms of that form must chooses $n-k$ many x's, and k many y's. Because just choosing the k many y's specifies the rest to be x's, there are $C(n,k)$ such terms. ■

Example 4.21. What is the coefficient of $x^{13}y^7$ in the expansion of $(x-3y)^{20}$? We write $(x-3y)^{20}$ as $(x+(-3y))^{20}$ and apply the binomial theorem, which gives us the term $C(20,7)x^{13}(-3y)^7 = -3^7C(20,7)x^{13}y^7$.

If we substitute specific values for x and y, the binomial theorem gives us more combinatorial identities as corollaries.

Corollary 4.22. $\sum_{k=0}^{n} C(n,k) = 2^n$, *again.*

Proof. Simply write $2^n = (1+1)^n$ and expand using the binomial theorem. ■

Corollary 4.23. $\sum_{k=1}^{n}(-1)^{k+1}C(n,k) = 1.$

Proof. Expand $0 = 0^n = (1-1)^n$ using the binomial theorem:

$$\begin{aligned} 0 &= \sum_{k=0}^{n} C(n,k)1^{n-k}(-1)^k \\ &= C(n,0) + \sum_{k=1}^{n}(-1)^k C(n,k) \end{aligned}$$

Rearranging terms gives us

$$C(n,0) = -\sum_{k=1}^{n}(-1)^k C(n,k) = \sum_{k=1}^{n}(-1)^{k+1}C(n,k)$$

This proves the corollary since $C(n,0) = 1$. ■

4.4 Inclusion-Exclusion Principle

Some counting problems simply do not have a closed form solution (drats!). In this section we discuss an counting tool that also does not give a closed form solution. The inclusion-exclusion principle can been seen as a generalization of the sum rule.

Suppose there are $n(A)$ ways to perform task A and $n(B)$ ways to perform task B, how many ways are there to perform task A or B, if these tasks are *not* distinct? We can cast this as a set cardinality problem. Let X be the set of ways to perform A, and Y be the set of ways to perform B. Then:

$$|X \cup Y| = |X| + |Y| - |X \cap Y|$$

This can be observed using the Venn Diagram. The counting argument goes as follows: To count the number of ways to perform A or B ($|X \cup Y|$) we start by adding the number of ways to perform A (i.e., $|X|$) and the number of ways to perform B (i.e., $|Y|$). But if some of the ways to perform A and B are the same ($|X \cap Y|$), they have been counted twice, so we need to subtract those.

Example 4.24. How many positive integers ≤ 100 are multiples of either 2 or 5? Let A be the set of multiples of 2 and B be the set of multiples of 5. Then $|A| = 50$, $|B| = 20$, and $|A \cap B| = 10$ (since this is the number of multiples of 10). By the inclusion-exclusion principle, we have $50 + 20 - 10 = 60$ multiples of either 2 or 5.

What if there are more tasks? For three sets, we can still gleam from the Venn diagram that

$$|X \cup Y \cup Z| = |X| + |Y| + |Z| - |X \cap Y| - |X \cap Z| - |Y \cap Z| + |X \cap Y \cap Z|$$

More generally,

Theorem 4.25. *Let $A_1, \ldots, A_n$ be finite sets. Then,*

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{I, I \subseteq \{1,\ldots,n\}, |I|=k} \left|\bigcap_{i \in I} A_i\right| = \sum_{I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} \left|\bigcap_{i \in I} A_i\right|$$

Proof. Consider some $x \in \cup_i A_i$. We need to show that it gets counted exactly one in the RHS. Suppose that x is contained in exactly m of the starting sets (A_1 to A_n), $1 \leq m \leq n$. Then for each $k \leq m$, x appears in $C(m,k)$ many k-way intersections (that is, if we look at $\left|\bigcap_{i \in I} A_i\right|$ for all $|I| = k$, x appears in $C(m,k)$ many terms). Therefore, the number of times x gets counted by the inclusion-exclusion formula is exactly

$$\sum_{k=1}^{m} (-1)^{k+1} C(m,k)$$

and this is 1 by Corollary 4.23. ∎

Example 4.26. How many onto functions are there from a set A with n elements to a set B with $m \leq n$ elements? We start by computing the number of functions that are not onto. Let A_i be the set of functions that miss the i^{th} element of B (i.e., does not have the i^{th} element of B in its range). $\cup_{i=1}^{m} A_i$ is then the set of functions that are not onto. By the inclusion exclusion principle, we have:

$$\left|\bigcup_{i=1}^{m} A_i\right| = \sum_{k=1}^{m} (-1)^{k+1} \sum_{I \subseteq \{1,\ldots,m\}, |I|=k} \left|\bigcap_{i \in I} A_i\right|$$

For any k and I with $|I| = k$, observe that $\cap_{i \in I} A_i$ is the set of functions that miss a particular set of k elements, therefore

$$\left|\bigcap_{i \in I} A_i\right| = (m-k)^n$$

Also observe that there are exactly $C(m,k)$ many different I's of size k. Using these two facts, we have

$$\begin{aligned}\left|\bigcup_{i=1}^{m} A_i\right| &= \sum_{k=1}^{m} (-1)^{k+1} \sum_{I \subseteq \{1,\ldots,m\}, |I|=k} \left|\bigcap_{i \in I} A_i\right| \\ &= \sum_{k=1}^{m} (-1)^{k+1} C(m,k)(m-k)^n\end{aligned}$$

Finally, to count the number of onto functions, we take all possible functions (m^n many) and subtract that functions that are not onto:

$$m^n - \sum_{k=1}^{m}(-1)^{k+1}C(m,k)(m-k)^n = m^n + \sum_{k=1}^{m}(-1)^k C(m,k)(m-k)^n$$
$$= \sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n$$

where the last step relies on $(-1)^k C(m,k)(m-k)^n = m^n$ when $k = 0$. This final expression $\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n$ is closely related to the *Sterling number of the second kind*, another counting function (similar to $C(n,k)$) that is out of the scope of this course.

4.5 Pigeonhole Principle

In this final section, we cover the pigeonhole principle: a proof technique that relies on counting. The principle says that if we place $k+1$ or more pigeons into k pigeon holes, then at least one pigeon hole contains 2 or more pigeons. For example, in a group of 367 people, at least two people must have the same birthday (since there are a total of 366 possible birthdays). More generally, we have

Lemma 4.27 (Pigeonhole Principle)**.** *If we place n (or more) pigeons into k pigeon holes, then at least one box contains $\lceil n/k \rceil$ or more pigeons.*

Proof. Assume the contrary that every pigeon hole contains $\leq \lceil n/k \rceil - 1 < n/k$ many pigeons. Then the total number of pigeons among the pigeon holes would be *strictly* less than $k(n/k) = n$, a contradiction. ■

Example 4.28. In a group of 800 people, there are at least $\lceil 800/366 \rceil = 3$ people with the same birthday.

Chapter 5

Probability

"... the chances of survival are 725 ... to 1."
– C-3PO

Originally motivated by gambling, the study of probability is now fundamental to a wide variety of subjects, including social behavior (e.g., economics and game theory) and physical laws (e.g., quantum mechanics and radioactive decay). In computer science, it is essential for studying randomized algorithms (e.g., randomized quick sort, primality testing), average-case problems (e.g., spam detection), and cryptography.

What is probability? What does it mean that that a fair coin toss comes up heads with probability 50%? One interpretation is *Bayesian*: "50%" is a statement of our beliefs, and how much we are willing to bet on *one* coin toss. Another interpretation is more experimental: "50%" means that if we toss the coin 10 million times, it will come up heads in roughly 5 million tosses. Regardless of how we view probability, this chapter introduces the mathematical formalization of probability, accompanied with useful analytical tools to go with the formalization.

5.1 Probability Spaces

In this class, we focus on *discrete* probability spaces.[1]

Definition 5.1 (Probability Space)**.** A **probability space** is a pair (S, f) where S is a countable set called the **sample space**, and $f : S \to [0, 1]$ is called the **probability mass function**.[2] Additionally, f satisfies the property $\sum_{x \in S} f(x) = 1$.

Intuitively, the sample space S corresponds to the set of possible states that the world could be in, and the probability mass function f assigns a

[1] Without formally defining this term, we refer to random processes whose outcomes are discrete, such as dice rolls, as opposed to picking a uniformly random real number from zero to one.

[2] By $[0, 1]$ we mean the real interval $\{x \mid 0 \leq x \leq 1\}$

probability from 0 to 1 to each of these states. To model our conventional notion of probability, we require that the total probability assigned by f to all possible states should sum up to 1.

Definition 5.2 (Event)**.** Given a probability space (S, f), an **event** is simply a subset of S. The probability of an event E, denoted by $\Pr_{(S,f)}[E] = \Pr[E]$, is defined to be $\sum_{x \in E} f(x)$. In particular, the event that includes "everything", $E = S$, has probability $\Pr[S] = 1$.

Even though events and probabilities are not well-defined without a probability space (e.g., see the quote of the chapter), by convention, we often omit S and f in our statements when they are clear from context.

Example 5.3. Consider rolling a regular 6-sided die. The sample space is $S = \{1, 2, 3, 4, 5, 6\}$, and the probability mass function is constant: $f(x) = 1/6$ for all $x \in S$. The event of an even roll is $E = \{2, 4, 6\}$, and this occurs with probability

$$\Pr[E] = \sum_{x \in \{2,4,6\}} f(x) = \frac{1}{2}$$

The probability mass function used in the above example has a (popular) property: it assigns equal probability to all the elements in the sample space.

Equiprobable Probability Spaces

Definition 5.4. A probability space (S, f) is equiprobable if f is constant, i.e., there exists a constant ε such that $f(x) = \varepsilon$ for all $x \in S$. In this case we call f an equiprobable mass function.

The most common examples of probability spaces, such as rolling a dice, flipping a coin and dealing a deck of (well-shuffled) cards, are all equiprobable. The next theorem reveals a common structure (and limitation) among equiprobable probability spaces.

Theorem 5.5. *Let (S, f) be an equiprobable probability space. Then S is finite, f takes on the constant value $1/|S|$, and the probability of an event $E \subseteq S$ is $|E|/|S|$.*

In other words, calculating probabilities under an equiprobable probability space is just counting.

Proof. Let f take on the constant value ε. First note that $\varepsilon \neq 0$, because it would force $\sum_{x \in S} f(x) = 0$, violating the definition of mass functions. Next note that S cannot be infinite, because it would force $\sum_{x \in S} f(x) = \sum_{x \in S} \varepsilon = \infty$, again violating the definition of mass functions.

Knowing that S is finite, we can then deduce that:

$$1 = \sum_{x \in S} f(x) = \sum_{x \in S} \varepsilon = |S|\varepsilon$$
$$\Rightarrow \quad |S| = \frac{1}{\varepsilon}$$

It follows that the probability of an event E is

$$\Pr[E] = \sum_{x \in E} \varepsilon = \sum_{x \in E} \frac{1}{|S|} = \frac{|E|}{|S|}$$ ■

Example 5.6. What is the probability that a random hand of five cards in poker is a full house? We have previously counted the number of possible five-card hands and the number of possible full houses (Example 4.12 and 4.13). Since each hand is equally likely (i.e., we are dealing with an equiprobable probability space), the probability of a full house is:

$$\frac{\#\text{ of possible full houses}}{\#\text{ of possible hands}} = \frac{C(13,1)C(4,3)C(12,1)C(4,2)}{C(52,5)} \approx 0.144\%$$

Theorem 5.5 highlights two limitations of equiprobable probability spaces. Firstly, the sample space must be finite; as we will soon see, some natural random processes require an infinite sample space. Secondly, the (equally assigned) probability of any event must be rational. That is, we cannot have an event that occurs with probability $1/\sqrt{2}$; this is required, for example, to formalize game theory in a satisfactory way. Despite these limitations, most of the time, we will deal with equiprobable probability spaces.

Infinite Sample Spaces

How might we construct a probability mass function for an infinite space? For example, how might one pick a random positive integer ($S = \mathbb{N}^+$)? We illustrate some possibilities (and subtleties) with the following examples.

Example 5.7. We may have the probability space $(\mathbb{N}^+, f)$ where $f(n) = 1/2^n$. This corresponds with the following experiment: how many coin tosses does it take for a head to come up?[3] We expect this to be a well-defined probability space since it corresponds to a natural random process. But to make sure, we verify that $\sum_{n \in \mathbb{N}^+} 1/2^n = 1$.[4]

[3] To see this, observe that in order for the first head to occur on the n^{th} toss, we must have the *unique* sequence of tosses that start with $n-1$ tails and end with a head. On the other hand, there are 2^n equally-probable sequences of n coin tosses. This gives probability $1/2^n$.

[4] One way to compute the sum is to observe that it is a converging geometric series. More directly, let $S = 1/2 + 1/4 + \cdots$, and observe that $S = 2S - S = (1 + 1/2 + 1/4 + \cdots) - (1/2 + 1/4 + \cdots) = 1$.

Example 5.8. Perhaps at a whim, we want to pick the positive integer n with probability proportional to $1/n^2$. In this case we need to *normalize* the probability. Knowing that $\sum_{n\in\mathbb{N}^+} 1/n^2 = \pi^2/6$,[5] we may assign $f(n) = (6/\pi^2)(1/n^2)$, so that $\sum_{n\in\mathbb{N}^+} f(n) = 1$.

Example 5.9. Suppose now we wish to pick the positive integer n with probability proportional to $1/n$. This time we are bound to fail, since the series $1 + 1/2 + 1/3 + \cdots$ diverges (approaches ∞), and cannot be normalized.

Probabilities

Now that probability spaces are defined, we give a few basic properties of probability:

Claim 5.10. *If A and B are disjoint events ($A \cap B = \emptyset$) then* $\Pr[A \cup B] = \Pr[A] + \Pr[B]$.

Proof. By definition,

$$\begin{aligned}\Pr[A \cup B] &= \sum_{x\in A\cup B} f(x) \\ &= \sum_{x\in A} f(x) + \sum_{x\in B} f(x) \qquad \text{since } A \text{ and } B \text{ are disjoint} \\ &= \Pr[A] + \Pr[B]\end{aligned}$$

■

Corollary 5.11. *For any event E,* $\Pr[\overline{E}] = 1 - \Pr[E]$.

Proof. This follows directly from Claim 5.10, $\overline{E} \cup E = S$, and $\overline{E} \cap E = \emptyset$. ■

When events are not disjoint, we instead have the following generalization of the inclusion-exclusion principle.

Claim 5.12. *Given events A and B,* $\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$.

Proof. First observe that $A \cup B = (A - B) \cup (B - A) \cup (A \cap B)$ and that all the terms on the RHS are disjoint. Therefore

$$\Pr[A \cup B] = \Pr[A - B] + \Pr[B - A] + \Pr[A \cap B] \tag{5.1}$$

Similarly, we have

$$\Pr[A] = \Pr[A - B] + \Pr[A \cap B] \tag{5.2}$$
$$\Pr[B] = \Pr[B - A] + \Pr[A \cap B] \tag{5.3}$$

[5] This is the Basel problem, first solved by Euler.

because, say A is the disjoint union of $A - B$ and $A \cap B$. Substituting (5.2) and (5.3) into (5.1) gives

$$\begin{aligned}\Pr[A \cup B] &= \Pr[A - B] + \Pr[B - A] + \Pr[A \cap B] \\ &= (\Pr[A] - \Pr[A \cap B]) + (\Pr[B] - \Pr[A \cap B]) + \Pr[A \cap B] \\ &= \Pr[A] + \Pr[B] - \Pr[A \cap B]\end{aligned}$$ ■

We remark that given an equiprobable probability space, Claim 5.12 is exactly equivalent to the inclusion-exclusion principle. An easy corollary of Claim 5.12 is the *union bound*.

Corollary 5.13 (Union Bound)**.** *Given events A and B,* $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$. *In general, given events $A_1 \ldots, A_n$,*

$$\Pr\left[\bigcup_i A_i\right] \leq \sum_i \Pr[A_i]$$

5.2 Conditional Probability and Independence

Let us continue with the interpretation that probabilities represent our beliefs on the state of the world. How does knowing that one event has occurred affect our beliefs on the probability of another event? E.g., if it is cloudy instead of sunny, then it is more likely to rain. Perhaps some events are *independent* and do not affect each other. E.g., we believe the result of a fair coin-toss does not depend on the result of previous tosses. In this section we capture these notions with the study of conditional probability and independence.

Conditional Probability

Suppose after receiving a random 5-card hand dealt from a standard 52-card deck, we are told that the hand contains "at least a pair" (that is, at least two of the cards have the same rank). How do we calculate the probability of a full-house given this extra information? Consider the following thought process:

- Start with the original probability space of containing all 5-card hands, pair or no pair.
- To take advantage of our new information, eliminate all hands that do not contain a pair.
- Re-normalize the probability among the remaining hands (that contain at least a pair).

Motivate by this line of reasoning, we define conditional probability as follows:

Definition 5.14. Let A and B be events, and let $\Pr[B] \neq 0$. The conditional probability of A, conditioned on B, denoted by $\Pr[A \mid B]$, is defined as

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

In the case of an equiprobable probability space, we have

$$\Pr[A \mid B] = |A \cap B|/|B|$$

because the probability of an event is proportional to the cardinality of the event.

Example 5.15 (Second Ace Puzzle)**.** Suppose we have a deck of four cards: $\{A\spadesuit, 2\spadesuit, A\heartsuit, 2\heartsuit\}$. After being dealt two random cards, facing down, the dealer tells us that we have at least one ace in our hand. What is the probability that our hand has both aces? That is, what is Pr[two aces | at least one ace]?

Because we do not care about the order in which the cards were dealt, we have an equiprobable space with 6 outcomes:

$$\{A\spadesuit 2\spadesuit, A\spadesuit A\heartsuit, A\spadesuit 2\heartsuit, 2\spadesuit A\heartsuit, 2\spadesuit 2\heartsuit, A\heartsuit 2\heartsuit\}$$

If we look closely, five of the outcomes contain at least one ace, while only one outcome has both aces. Therefore Pr[two aces | at least one ace] = 1/5.

Now what if the dealer tells us that we have the ace of spades ($A\spadesuit$) in our hand? Now Pr[two aces | has ace of spades] = 1/3. It might seem strange that the probability of two aces has gone up; why should finding out the suit of the ace we have increase our chances? The intuition is that by finding out the suit of our ace and *knowing* that the suit is spades, we can eliminate many more hands that are not two aces.

Example 5.16 (Second-Child Problem)**.** Let us assume that a child is equally likely to be a boy or a girl. A friend of yours has two children but you don't know their sexes. Consider the following two cases. A boy walks in the door and your friend says,

"This is my child." (5.4)

Or, a boy walks in the door and your friend says,

"This is my older child." (5.5)

What is the probability that both children are boys?

If we order the children by age, we again have a equiprobable probability space[6]. The four outcomes are { boy-boy, boy-girl, girl-boy, girl-girl }. Therefore,

$$\Pr[\text{ two boys} \mid (5.4)] = 1/3$$
$$\Pr[\text{ two boys} \mid (5.5)] = 1/2$$

[6] If we do not order the children, then our sample space could be {two boys, one boy and one girl, two girls}, with probability 1/4, 1/2, and 1/4 respectively. This probability space is suitable for case (5.4), but not (5.5)

Now suppose that we know exactly one of the children plays the cello. If a boy walks in the door and your friend says,

"He is the one who plays the cello."

then we are in same case as (5.5) (instead of ordering by age, we order the children according to who plays the cello).

One more food for thought. What if we know that at least one of the children plays the cello? Now what is the probability that both children are boys, if a boy walks in, and start playing the cello? To calculate this probability, first we need to enlarge the sample space; each child, in addition to being a boy or a girl, either plays the cello or does not. Next we need to specify a probability mass function on this space. This is where we get stuck, since we need additional information to define the probability mass function. E.g., what is the probability that both children plays the cello, v.s. only one child plays the cello?

Independence

By defining conditional probability, we modeled how the occurrence of one event can affect the probability of another event. An equally interesting concept is *independence*, where a set of events *do not* affect each other.

Definition 5.17 (Independence)**.** A sequence of events $A_1, \ldots, A_n$ are (mutually) independent[7] if and only if for every subset of these events, $A_{i_1}, \ldots, A_{i_k}$,

$$\Pr[A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}] = \Pr[A_{i_1}] \cdot \Pr[A_{i_2}] \cdots \Pr[A_{i_k}]$$

If there are just two events, A and B, then they are independent if and only if $\Pr[A \cap B] = \Pr[A] \Pr[B]$. The following claim gives justification to the definition of independence.

Claim 5.18. *If A and B are independent events and $\Pr[B] \neq 0$, then $\Pr[A \mid B] = \Pr[A]$. In other words, conditioning on B does not change the probability of A.*

Proof.

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]} = \frac{\Pr[A] \Pr[B]}{\Pr[B]} = \Pr[A] \qquad \blacksquare$$

The following claim should also hold according to our intuition of independence:

[7] A related notion that we do not cover in this class is pair-wise independence. A sequence of events $A_1, \ldots, A_n$ are pair-wise independent if and only if for every *pair* of events in the sequence, A_{i_1}, A_{i_2}, we have $\Pr[A_{i_1} \cap A_{i_2}] = \Pr[A_{i_1}] \Pr[A_{i_2}]$. Pair-wise independence is a weaker requirement than (mutual) independence, and is therefore easier to achieve in applications.

Claim 5.19. *If A and B are independent events, then A and $\overline{B}$ are also independent events. In other words, if A is independent of the occurrence of B, then it is also independent of the "non-occurrence" of B.*

Proof.

$$\begin{aligned}
\Pr[A \cap \overline{B}] &= \Pr[A] - \Pr[A \cap B] && \text{Claim 5.10} \\
&= \Pr[A] - \Pr[A]\Pr[B] && \text{by independence} \\
&= \Pr[A](1 - \Pr[B]) = \Pr[A]\Pr[\overline{B}] && \text{Corollary 5.11}
\end{aligned}$$ ■

A common use of independence is to predict the outcome of n coin tosses, or more generally, the outcome of n independent Bernoulli trials (for now think of a Bernoulli trial with success probability p as a biased coin toss that comes up "success" with probability p and "failure" with probability $1 - p$).

Theorem 5.20. *The probability of having exactly k successes in n independent Bernoulli trials with success probability p is $C(n, k)p^k(1 - p)^{n-k}$.*

Proof. If we denote success by S and failure by F, then our probability space is the set of n-character strings containing the letters S and F (e.g., $SFF \cdots F$ denotes the out come that the first Bernoulli trial is successful, while all the rest failed). Using our counting tools, we know that number of such strings with exactly k occurrences of S (success) is $C(n, k)$. Each of those strings occur with probability $p^k(1 - p)^{n-k}$ due to independence. ■

Bayes' Rule

Suppose that we have a test against a rare disease that affects only 0.3% of the population, and that the test is 99% effective (i.e., if a person has the disease the test says YES with probability 0.99, and otherwise it says NO with probability 0.99). If a random person in the populous tested positive, what is the probability that he has the disease? The answer is not 0.99. Indeed, this is an exercise in conditional probability: what are the chances that a random person has the rare disease, given the occurrence of the event that he tested positive?

We start with with some preliminaries.

Claim 5.21. *Let $A_1, \ldots, A_n$ be disjoint events with non-zero probability such that $\bigcup_i A_i = S$ (i.e., the events are exhaustive; the events partition the sample space S). Let B be an event. Then $\Pr[B] = \sum_{i=1}^{n} \Pr[B \mid A_i]Pr[A_i]$*

Proof. By definition $\Pr[B \mid A_i] = \Pr[B \cap A_i] / \Pr[A_i]$, and so the RHS evaluates to

$$\sum_{i=1}^{n} \Pr[B \cap A_i]$$

Since $A_1, \ldots, A_n$ are disjoint it follows that the events $B \cap A_1, \ldots, B \cap A_n$ are also disjoint. Therefore

$$\sum_{i=1}^{n} \Pr[B \cap A_i] = \Pr\left[\bigcup_{i=1}^{n} B \cap A_i\right] = \Pr\left[B \cap \bigcup_{i=1}^{n} A_i\right] = \Pr\left[B \cap S\right] = \Pr[B]$$

■

Theorem 5.22 (Bayes' Rule)**.** *Let A and B be events with non-zero probability. Then:*

$$\Pr[B \mid A] = \frac{\Pr[A \mid B] \Pr[B]}{\Pr[A]}$$

Proof. Multiply both sides by $\Pr[A]$. Now by definition of conditional prob, both sides equal:

$$\Pr[B \mid A] \Pr[A] = \Pr[A \cap B] = \Pr[A \mid B] \Pr[B]$$ ■

A remark on notation: the symbol for conditioning "|" is similar to that of division. While $\Pr[A \mid B]$ is *definitely not* $\Pr[A]/\Pr[B]$, it does have the form "stuff/ $\Pr[B]$". In this sense, the above two proofs are basically "multiply by the denominator".

Sometimes we expand the statement of Bayes' Rule is with Claim 5.21:

Corollary 5.23 (Bayes' Rule Expanded)**.** *Let A and B be events with non-zero probability. Then:*

$$\Pr[B \mid A] = \frac{\Pr[A \mid B] \Pr[B]}{\Pr[B] \Pr[A \mid B] + \Pr[\overline{B}] \Pr[A \mid \overline{B}]}$$

Proof. We apply Claim 5.21, using that B and $\overline{B}$ are disjoint and $B \cup \overline{B} = S$. ■

We return to our original question of testing for rare diseases. Let's consider the sample space $S = \{(t, d) \mid t \in \{0, 1\}, d \in \{0, 1\}\}$, where t represents the outcome of the test on a random person in the populous, and d represents whether the same person carries the disease or not. Let D be event that a randomly drawn person has the disease ($d = 1$), and T be the event that a randomly drawn person tests positive ($t = 1$).

We know that $\Pr[D] = 0.003$ (because 0.3% of the population has the disease). We also know that $\Pr[T \mid D] = 0.99$ and $\Pr[T \mid \overline{D}] = 0.01$ (because the test is 99% effective). Using Bayes' rule, we can now calculate the probability that a random person, who tested positive, actually has the disease:

$$\begin{aligned}\Pr[D \mid T] &= \frac{\Pr[T \mid D] \Pr[D]}{(\Pr[D] \Pr[T \mid D] + \Pr[\overline{D}] \Pr[T \mid \overline{D}])} \\ &= \frac{.99 * .003}{.003 * .99 + .997 * .01} = 0.23\end{aligned}$$

Notice that 23%, while significant, is a far cry from 99% (the effectiveness of the test). This final probability can vary if we have a different *prior* (initial belief). For example, if a random patient has other medical conditions that raises the probability of contracting the disease up to 10%, then the final probability of having the disease, given a positive test, raises to 92%.

Conditional Independence

Bayes' rule shows us how to update our beliefs when we receive new information. What if we receive multiple signals at once? How do we compute $\Pr[A \mid B_1 \cap B_2]$? First we need the notion of conditional independence.

Definition 5.24 (Conditional Independence)**.** A sequence of events $B_1, \ldots, B_n$ are conditionally independent given event A if and only if for every subset of the sequence of events, $B_{i_1}, \ldots, B_{i_k}$,

$$\Pr\left[\bigcap_k B_{i_k} \mid A\right] = \prod_k \Pr[B_{i_k} \mid A]$$

In other words, given that the event A has occurred, then the events $B_1, \ldots, B_n$ are independent.

When there are only two events, B_1 and B_2, they are conditionally independent given event A if and only if $\Pr[B_1 \cap B_2 \mid A] = \Pr[B_1 \mid A] \Pr[B_2 \mid A]$. The notion of conditional independence is somewhat fickle, illustrated by the following examples:

Independence does not imply conditional independence. Suppose we toss a fair coin twice; let H_1 and H_2 be the event that the first and second coin tosses come up heads, respectively. Then H_1 and H_2 are independent:

$$\Pr[H_1] \Pr[H_2] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} = \Pr[H_1 \cap H_2]$$

However, if we are told that the at least one of the coin tosses came up tails (call this event T), then H_1 and H_2 are no longer independent given T:

$$\Pr[H_1 \mid T] \Pr[H_2 \mid T] = \frac{1}{3} \cdot \frac{1}{3} \neq 0 = \Pr[H_1 \cap H_2 \mid T]$$

Conditional independence does not imply independence. Suppose we have two coins, one is heavily biased towards heads, and the other one is heavily biased towards tails (say with probability 0.99). First we choose a coin at random; let B_H be the event that we choose the coin that is biased towards heads. Next we toss the chosen coin twice; let H_1 and H_2 be the event that the first and second coin tosses come up heads, respectively. Then, given that we chose the coin biased towards heads (the event B_H), H_1 and H_2 are independent:

$$\Pr[H_1 \mid F] \Pr[H_2 \mid F] = 0.99 \cdot 0.99 = 0.99^2 = \Pr[H_1 \cap H_2 \mid F]$$

However, H_1 and H_2 are not independent, since if the first toss came up heads, it is most likely that we chose the coin that is biased towards heads, and so the second toss will come up heads as well. Actually probabilities are:

$$\Pr[H_1]\Pr[H_2] = \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4} \neq \Pr[H_1 \cap H_2] = 0.5(0.99^2) + 0.5(0.01^2) \approx 0.5$$

Independence conditioned on event A does not imply independence conditioned on the complement event $\overline{A}$. Consider twins. Let G_1 and G_2 be the event that the first and second child is a girl, respectively. Let A be the event that the twins are fraternal (non-identical). Then given event A, it is reasonable to assume that G_1 and G_2 are independent. On the other hand, given event $\overline{A}$ (identical twins), then G_1 and G_2 are most certainly dependent since identical twins must both be boys or both be girls.

Let us return to the question of computing $\Pr[A \mid B_1 \cap B_2]$. If we assume that the signals B_1 and B_2 are independent when conditioned on A, and also independent when conditioned on $\overline{A}$, then:

$$\begin{aligned}
&\Pr[A \mid B_1 \cap B_2] \\
&= \frac{\Pr[B_1 \cap B_2 \mid A]\Pr[A]}{\Pr[A]\Pr[B_1 \cap B_2 \mid A] + \Pr[\overline{A}]\Pr[B_1 \cap B_2 \mid \overline{A}]} \\
&= \frac{\Pr[B_1 \mid A]\Pr[B_2 \mid A]\Pr[A]}{\Pr[A]\Pr[B_1 \mid A]\Pr[B_2 \mid A] + \Pr[\overline{A}]\Pr[B_1 \mid \overline{A}]\Pr[B_2 \mid \overline{A}]}
\end{aligned}$$

In general, given signals $B_1, \ldots, B_n$ that are conditionally independent given A and conditionally independent given $\overline{A}$, we have

$$\Pr\left[A \mid \bigcap_i B_i\right] = \frac{\Pr[A]\prod_i \Pr[B_i \mid A]}{\Pr[A]\prod_i \Pr[B_i \mid A] + \Pr[\overline{A}]\prod_i \Pr\left[B_i \mid \overline{A}\right]}$$

Application: Spam Detection

Using "training data" (e-mails classified as spam or not by hand), we can estimate the probability that a message contains a certain string conditioned on being spam (or not), e.g., Pr["viagra" | spam], Pr["viagra" | not spam]. We can also estimate the chance that a random e-mail is spam, i.e., Pr[spam] (this is about 80% in real life, although most spam detectors are "unbiased" and assume Pr[spam] = 50% to make calculations nicer).

By choosing a diverse set of keywords, say $W_1, \ldots, W_n$, and assuming that the occurrence of these keywords are conditionally independent given a spam message or given a non-spam e-mail, we can use Bayes' rule to estimate the chance that an e-mail is spam based on the words it contains (we have simplified

the expression assuming Pr[spam] = Pr[not spam] = 0.5):

$$\Pr\left[\text{spam} \mid \bigcap_i W_i\right] = \frac{\prod_i \Pr\left[W_i \mid \text{spam}\right]}{\prod_i \Pr\left[W_i \mid \text{spam}\right] + \prod_i \Pr\left[W_i \mid \text{not spam}\right]}$$

5.3 Random Variables

We use *events* to express whether a particular class of outcomes has occurred or not. Sometimes we want to express more: for example, after 100 fair coin tosses, we want to study how many coin tosses were heads (instead of focusing on just one event, say, that there were 50 coin tosses). This takes us to the definition of *random variables*.

Definition 5.25. A random variable X on a probability space (S, f) is a function from the sample space to the real numbers $X : S \to \mathbb{R}$.

Back to the example of 100 coin tosses, given any outcome of the experiment $s \in S$, we would define $X(s)$ to be the number of heads that occurred in that outcome.

Definition 5.26. Given a random variable X on probability space (S, f), we can consider a new probability space (S', f_X) where the sample space is the range of X, $S' = \{X(s) \mid s \in S\}$, and the probability mass function is extended from f, $f_X(x) = \Pr_{S,f}[\{x \mid X(s) = x\}]$. We call f_X the *probability distribution* or the *probability density function* of the random variable X. Similarly defined, the *cumulative distribution* or the *cumulative density function* of the random variable X is $F_X(x) = \Pr_{S,f}[\{x \mid X(s) \leq x\}]$.

Example 5.27. Suppose we toss two 6-sided dice. The sample space would be pairs of outcomes, $S = \{(i, j) \mid i, j \in \{1, \ldots, 6\}\}$, and the probability mass function is equiprobable. Consider the random variables, $X_1(i, j) = i$, $X_2(i, j) = j$ and $X(i, j) = i + j$. These random variables denotes the outcome of the first die, the outcome of the second die, and the some of the two die, respectively. The probability density function of X would take values:

$$\begin{aligned}
f_X(1) &= 0 \\
f_X(2) &= \Pr[(1,1)] = 1/36 \\
f_X(3) &= \Pr[(1,2),(2,1)] = 2/36 \\
&\vdots \\
f_X(6) &= \Pr[(1,5),(2,3),\ldots,(3,1)] = 5/36 \\
f_X(7) &= \Pr[(1,6)..(6,1)] = 6/36 \\
f_X(8) &= \Pr[(2,6),(3,5),\ldots,(6,2)] = 5/36 = f_X(6) \\
&\vdots \\
f_X(12) &= 1/36
\end{aligned}$$

And the cumulative density function of X would be:

$$\begin{aligned} F_X(2) &= 1/36 \\ F_X(3) &= 1/36 + 2/36 = 1/12 \\ &\vdots \\ F_X(12) &= 1 \end{aligned}$$

Notation Regarding Random Variables

We can describe events by applying predicates to random variables (e.g., the event that X, the number of heads, is equal to 50). We often use a short-hand notation (in which we treat random variables as if they are real numbers), as demonstrated in the following examples. Let X and Y be random variables:

$$\begin{aligned} X = 50 \text{ is the event } \{s \in S \mid X(s) = 50\} \\ Y \leq X \text{ is the event } \{s \in S \mid Y(s) \leq X(s)\} \end{aligned}$$

Using this notation, we may define the probability density function of a random variable X as $f_X(x) = \Pr[X = x]$, and the cumulative density function as $F_X(x) = \Pr[X \leq x]$.

In a similar vain, we can define new random variables from existing random variables. In Example 5.27, we can write $X = X_1 + X_2$, to mean that for any $s \in S$, $X(s) = X_1(s) + X_2(s)$ (again, the notation treats, X, X_1 and X_2 as if the are real numbers).

Independent Random Variables

The intuition behind independent random variables is just like that of events: the value of one random variable should not affect the value of another independent random variable.

Definition 5.28. A sequence of random variables $X_1, X_2, \ldots, X_n$ are (mutually) independent if for every subset $X_{i_1}, \ldots, X_{i_k}$ and for any real numbers $x_1, x_2, \ldots, x_k$, the events $X_1 = x_{i_1}, X_2 = x_{i_2}, \ldots, X_{i_k} = x_k$ are (mutually) independent.

In the case of two random variables X and Y, they are independent if and only if for all real values x and y, $\Pr[X = x \cap X = y] = \Pr[X = x] \Pr[Y = y]$.

As mentioned before, a common use of independence is to model the outcome of consecutive coin tosses. This time we model it as the sum of independent random variables. Consider a biased coin that comes up heads with probability p. Define $X = 1$ if the coin comes up heads and $X = 0$ if the coin comes up tails; then X is called the Bernoulli random variable (with probability p). Suppose now we toss this biased coin n times, and let Y be the random variable that denotes the total number of occurrence of heads.[8] We can view

[8] Just for fun, we can calculate the density function and cumulative density function of Y. By Theorem 5.20, $f_Y(k) = C(n,k)p^k(1-p)^{n-k}$, and $F_Y(k) = \sum_{i=0}^{k} C(n,i)p^i(1-p)^{n-i}$.

Y as a sum of independent random variables, $\sum_{i=1}^{n} X_i$, where X_i is a Bernoulli random variable with probability p that represents the outcome of the i^{th} toss. We leave it as an exercise to show that the random variables $X_1, \ldots, X_n$ are indeed independent.

5.4 Expectatation

Given a random variable defined on a probability space, what is its "average" value? Naturally, we need to weigh things according to the probability that the random variable takes on each value.

Definition 5.29. Given a random variable X defined over a probability space (S, f), we define the expectation of X to be

$$\mathbb{E}[X] = \sum_{x \in \text{ range of } X} \Pr[X = x] \cdot x = \sum_{x \in \text{ range of } X} f_X(x) \cdot x$$

An alternative but equivalent definition is

$$\mathbb{E}[X] = \sum_{s \in S} f(s) X(s)$$

These definitions are equivalent because:

$$\begin{aligned}
&\sum_{x \in \text{ range of } X} \Pr[X = x] \cdot x \\
&= \sum_{x \in \text{ range of } X} \sum_{s \in (X = x)} f(s) \cdot x && \text{Expanding } \Pr[X = x], \text{ and recall that } X = x \text{ is the set } \{s \mid X(s) = x\} \\
&= \sum_{x \in \text{ range of } X} \sum_{s \in (X = x)} f(s) \cdot X(s) && \text{Replacing } x \text{ with } X(s) \\
&= \sum_{s \in S} f(s) X(s) && \text{the events } X = x \text{ partitions } S \text{ when } x \text{ ranges over the range of } X
\end{aligned}$$

The following simple fact can be shown with a similar argument:

Claim 5.30. *Given a random variable X and a function $g : \mathbb{R} \to \mathbb{R}$,*

$$\mathbb{E}[g(X)] = \sum_{x \in \textit{ range of } X} \Pr[X = x] g(x)$$

Proof.

$$\begin{aligned}
&\sum_{x\in \text{ range of } X} \Pr[X=x]g(x)\\
=&\sum_{x\in \text{ range of } X}\sum_{s\in(X=x)} f(s)g(x)\\
=&\sum_{x\in \text{ range of } X}\sum_{s\in(X=x)} f(s)g(X(s))\\
=&\sum_{s\in S} f(s)g(X(s)) = \mathbb{E}[g(X)]
\end{aligned}$$

■

Example 5.31. Suppose in a game, with probability 1/10 we are paid \$10, and with probability 9/10 we are paid \$2. What is our expected payment? The answer is

$$\frac{1}{10}\$10 + \frac{9}{10}\$2 = \$2.80$$

Example 5.32. Given a biased coin that ends up heads with probability p, how many tosses does it take for the coin to show heads, in expectation?

We may consider the state space $S = \{H, TH, TTH, TTTH, \dots\}$; these are possible results of a sequence of coin tosses that ends when we see the first head. Because each coin toss is independent, we define the probability mass function to be

$$f(T^iH) = f(i \text{ tails followed by a head}) = (1-p)^i p$$

We leave it as an exercise to show that f is a valid probabilistic mass function.[9]

Let X be the random variable that denote the number of coin tosses needed for heads to show up. Then $X(T^iH) = i+1$. The expectation of X is then

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=0}^{\infty}(i+1)p(1-p)^i\\
&= p\sum_{i=0}^{\infty}(i+1)(1-p)^i = p\frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

where we used the fact that $\sum_{i=0}^{\infty}(i+1)x^i = 1/(1-x)^2$ whenever $|x| < 1$.[10]

Application to Game Theory

In game theory, we assign a real number, called the *utility*, to each outcome in the sample space of a probabilistic game. We then assume that *rational* players

[9] Recall that an infinite geometric series with ratio $|x| < 1$ converges to $\sum_{i=0}^{\infty} x^i = 1/(1-x)$.

[10] To see this, let $S = \sum_{i=0}^{\infty}(i+1)x^i$, and observe that if $|x| < 1$, then $S(1-x)$ is a converging geometric series: $S(1-x) = S - xS = (x^0 + 2x^1 + 3x^2 + \cdots) - (x^1 + 2x^2 + \cdots) = (x^0 + x^1 + \cdots) = 1/(1-x)$.

make decisions that maximize their expected utility. For example, should we pay \$2 to participate in the game in Example 5.31? If we assume that our utility is exactly the amount of money that we earn, then

with probability 1/10 we get paid \$10 and gets utility 8
with probability 9/10 we get paid \$2 and gets utility 0

This gives a positive expected utility of 0.8, so we should play the game!

This reasoning of utility does not always explain human behavior though. Suppose there is a game that cost a thousand dollars to play. With one chance in a million, the reward is two billion dollars (!), but otherwise there is no reward. The expected utility is

$$\frac{1}{10^6}(2 \times 10^9 - 1000) + (1 - \frac{1}{10^6})(0 - 1000) = 1000$$

One expects to earn a thousand dollars from the game on average. Would you play it? Turns out many people are *risk-averse* and would turn down the game. After all, *except with one chance in a million*, you simply lose a thousand dollars. This example shows how expectation does not capture all the important features of a random variable, such as how likely does the random variable end up close to its expectation (in this case, the utility is either -1000 or two billion, not close to the expectation of 1000 at all).

In other instances, people are risk-seeking. Take yet another game that takes a dollar to play. This time, with one chance in a billion, the reward is a million dollars; otherwise there is no reward. The expected utility is

$$\frac{1}{10^9}(10^6 - 1) + (1 - \frac{1}{10^9})(0 - 1) = -0.999$$

Essentially, to play the game is to throw a dollar away. Would you play the game? Turns out many people do; this is called a lottery. Many people think losing a dollar will not change their life at all, but the chance of winning a million dollars is worth it, even if the chance is tiny. One way to explain this behavior within the utility framework is to say that perhaps earning or losing just a dollar is not worth 1 point in utility.

Linearity of Expectation

One nice property of expectation is that the expectation of the sum of random variables, is the sum of expectations. This can often simplify the calculation of expectation (or in applications, the estimation of expectation). More generally,

Theorem 5.33. *Let $X_1, \ldots, X_n$ be random variables, and $a_1, \ldots, a_n$ be real constants. Then*

$$\mathbb{E}\left[\sum_{i=1}^{n} a_i X_i\right] = \sum_{i=1}^{n} a_i \, \mathbb{E}[X_i]$$

Proof.

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n} a_i X_i\right] &= \sum_{s\in S} f(s) \sum_{i=1}^{n} a_i X_i(s) \\ &= \sum_{s\in S}\sum_{i=1}^{n} a_i f(s) X_i(s) \\ &= \sum_{i=1}^{n} a_i \sum_{s\in S} f(s) X_i(s) \\ &= \sum_{i=1}^{n} a_i \,\mathbb{E}[X_i]\end{aligned}$$

■

Example 5.34. If we make n tosses of a biased coin that ends up heads with probability p, what is the expected number of heads? Let $X_i = 1$ if the i^{th} toss is heads, and $X_i = 0$ otherwise. Then X_i is an independent Bernoulli random variable with probability p, and has expectation

$$\mathbb{E}[X_i] = p \cdot 1 + (1-p) \cdot 0 = p$$

The expected number of heads is then

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = np$$

Thus if the coin was fair, we would expect $(1/2)n$, half of the tosses, to be heads.

Variance

Consider the following two random variables:

1. X always take the constant value 1.
2. $Y = 0$ with probability $1 - 10^{-6}$, and $Y = 10^6$ with probability 10^{-6}.

Both X and Y has expectation 1, but they have very different distributions. To capture their differences, the *variance* of a random variable is introduced to capture how "spread out" is the random variable away from its expectation.

Definition 5.35. The variance of a random variable X is defined as

$$\text{Var}[X] = \mathbb{E}[(X\,\mathbb{E}[X])^2]$$

Intuitively, the term $(X - \mathbb{E}[X])^2$ measures the distance of X to its expectation. The term is squared to ensure that the distance is always positive (perhaps we could use absolute value, but it turns out defining variance with a square gives it much nicer properties).

Example 5.36. Let X be Bernoulli random variable with probability p. Then

$$\begin{aligned}\operatorname{Var}[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[(X-p)^2] \\ &= p(1-p)^2 + (1-p)(-p)^2 = (1-p)p(1-p+p) = (1-p)p\end{aligned}$$

Sometimes it is easier to calculate the variance using the following formula

Theorem 5.37. *Let X be a random variable. Then*

$$\operatorname{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

Proof.

$$\begin{aligned}\operatorname{Var}[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2] \\ &= \mathbb{E}[X^2 - (2\,\mathbb{E}[X])X + \mathbb{E}[X]^2] \\ &= \mathbb{E}[X^2] - (2\,\mathbb{E}[X])\,\mathbb{E}[X] + \mathbb{E}[X]^2 \qquad \text{by linearity of expectation} \\ &= \mathbb{E}[X^2] - \mathbb{E}[X]^2\end{aligned}$$

■

Example 5.38. Let X be Bernoulli random variable with probability p, and let us calculate its variance with the new formula:

$$\begin{aligned}\operatorname{Var}[X] &= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\ &= p(1)^2 + (1-p)(0)^2 - p^2 = p - p^2 = p(1-p)\end{aligned}$$

Chapter 6

Logic

"Logic will get you from A to B. Imagination will take you everywhere."
– Albert Einstein

Logic is a formal study of mathematics; it is the study of mathematic reasoning and proofs itself. In this chapter we cover two most basic forms of logic. In *propositional* logic, we consider basic conjunctives such as AND, OR, and NOT. In *first-order* logic, we additionally include tools to reason, for example, about "for all prime numbers" or "for some bijective function". There are many more logical systems studied by mathematicians that we do not cover (e.g., modal logic for reasoning about knowledge, or temporal logic for reasoning about time).

6.1 Propositional Logic

A formula in propositional logic consists of atoms and connectives. An **atom** is a primitive proposition, such as "it is raining in Ithaca", or "I open my umbrella"; we usually denote that atoms by capital letters (P, Q, R). The atoms are then connected by connectives, such as AND ($\wedge$), OR ($\vee$), NOT ($\neg$), implication ($\rightarrow$), iff ($\leftrightarrow$). An example of a formula is

$$(P \wedge Q) \rightarrow R$$

If P is the atom "it is raining in Ithaca", Q is the atom "I have an umbrella", and R is the atom "I open my umbrella", then the formula reads as:

If it is raining in Ithaca and I have an umbrella, then I open my umbrella.

Formally, we define **formulas** recursively:

- Every atom is a formula.
- If φ and ψ are formulas, then

  $$\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi$$

  are all valid formulas.

What does $P \vee Q \wedge R$ mean? Just like in arithmetic where multiplication has precedence over addition, here the order of precedence is: NOT ($\neg$), AND ($\wedge$), OR ($\vee$), implication ($\rightarrow$), equivalence ($\leftrightarrow$). The preferred way to disambiguate a formula, or course, is to use parenthesis (e.g., it is more clear and equivalent to write $P \vee (Q \wedge R)$).

Semantics of Propositional Logic

Here is how we interpret propositional logic. An atom can either be true (T or 1) or false (F or 0). This is specified by a truth assignment or, in logic jargon, an *interpretation* (e.g., an interpretation would specify whether today is really raining, or whether I have opened my umbrella). The connectives are functions from truth value(s) to a truth value; these functions are defined to reflect the meaning of the connectives' English names. The formal definition of these functions can be seen in a truth table in Figure 6.1.

φ	ψ	$\neg\varphi$	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \rightarrow \psi$	$\varphi \leftrightarrow \psi$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Figure 6.1: The truth table definition of the connectives NOT ($\neg$), AND ($\wedge$), OR ($\vee$), implication ($\rightarrow$), and equivalence ($\leftrightarrow$).

Most of the definitions are straightforward. NOT flips a truth value; AND outputs true iff both inputs are true, OR outputs true iff at least one of the inputs are true; equivalence outputs true iff both inputs have the same truth value. Implication ($\rightarrow$) may seem strange at first. $\varphi \rightarrow \psi$ is false only if φ is true, yet ψ is false. In particular, $\varphi \rightarrow \psi$ is true whenever φ is false, regardless of what ψ is. An example of this in English might be "if pigs fly, then I am the president of the United States"; this seems like a correct statement regardless of who says it since pigs don't fly in our world.[1]

Finally, we denote the truth value of a formula φ, *evaluated* on an interpretation I, by $\varphi[I]$. We define $\varphi[I]$ inductively:

- If φ is an atom P, then $\varphi[I]$ is the truth value assigned to P in the interpretation I.
- If $\varphi = \neg\rho$, then $\varphi[I] = \neg\rho[I]$ (using Table 6.1).
- If $\varphi = \rho_1 \wedge \rho_2$, then $\varphi[I] = \rho_1[I] \wedge \rho_2[I]$ (using Table 6.1). The value of $\varphi[I]$ is similarly defined if $\varphi = \rho_1 \vee \rho_2$, $\rho_1 \rightarrow \rho_2$ or $\rho_1 \leftrightarrow \rho_2$.

[1] A related notion, *counterfactuals*, is not captured by propositional implication. In the sentence "if pigs were to fly then they would have wings", the speaker knows that pigs do not fly, but wish to make a logical conclusion in an imaginary world where pigs do. Formalizing counterfactuals is still a topic of research in logic.

Given a formula φ, we call the mapping from interpretations to the truth value of φ (i.e., the mapping that takes I to $\varphi[I]$) the *truth table* of φ.

At this point, for convenience, we add the symbols T and F as special atoms that are always true or false, respectively. This does not add anything real substance to propositional logic since we can always replace T by "$P \vee \neg P$" (which always evaluates to true), and F by "$P \wedge \neg P$" (which always evaluates to false).

Equivalence of Formulas

We say that two formulas φ and ψ are equivalent (denoted $\varphi \equiv \psi$) if for all interpretations I, they evaluate to the same truth value (equivalently, if φ and ψ have the same truth table). How many possible truth tables are there over n atoms? Because each atom is either true or false, we have 2^n interpretations. A formula can evaluate to true or false on each of the interpretations, resulting in 2^{2^n} possible truth tables (essentially we are counting the number of functions of the form $\{0,1\}^n \to \{0,1\}$).

With such a large count of distinct (not equivalent) formulas, we may wonder is our propositional language rich enough to capture all of them? The answer is yes. The following example can be extended to show how AND, OR and NOT ($\wedge$, $\vee$ and $\neg$) can be used to capture any truth table. Suppose we want to capture the truth table for implication:

P	Q	$\varphi\ (= P \to Q)$
T	T	T
T	F	F
F	T	T
F	F	T

We find the rows where φ is true; for each such row we create an AND formula that is true iff P and Q takes on the value of that row, and then we OR these formulas together. That is:

$$\underbrace{(P \wedge Q)}_{\text{first row}} \vee \underbrace{(\neg P \wedge Q)}_{\text{third row}} \vee \underbrace{(\neg P \wedge \neg Q)}_{\text{fourth row}}$$

This can be simplified to the equivalent formula:

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

The equivalence

$$P \to Q \equiv \neg P \vee Q \tag{6.1}$$

is a very useful way to think about implication (and a very useful formula for manipulating logic expressions).

Finally, we remark that we do not need both OR and AND ($\vee$ and $\wedge$) to capture all truth tables. This follows from De Morgan's Laws:

$$\begin{aligned} \neg(\varphi \wedge \psi) &\equiv \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) &\equiv \neg\varphi \wedge \neg\psi \end{aligned} \tag{6.2}$$

Coupled with the (simple) equivalence $\neg\neg\varphi \equiv \varphi$, we can eliminate AND ($\wedge$), for example, using

$$\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$$

Satisfiability and Validity

Intuitively, a formula is satisfiable if it can be made true.

Definition 6.1 (Satisfiability)**.** We say that a truth assignment I **satisfies** a formula φ if $\varphi[I] = \mathsf{T}$; we write this as $I \models \varphi$. A formula φ is **satisfiable** if there exists a truth assignment I such that $I \models \varphi$; otherwise φ is unsatisfiable.

Even better, a formula is valid if it is always true.

Definition 6.2 (Validity)**.** A formula φ is *valid* (or a *tautology*) if for all a truth assignments I, $I \models \varphi$.

Example 6.3.

- $P \wedge Q$ is satisfiable.
- $P \wedge \neg P$ is unsatisfiable.
- $P \vee \neg P$ is valid.
- For a more complicated example, the following formula is valid.

  $$(P \to Q) \vee (Q \to P)$$

  To see why, note that it is equivalent to

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

  by (6.1), and clearly either $\neg P$ or P is true.

How do we check if a formula φ is valid, satisfiable or unsatisfiable? A simple way is to go over all possible truth assignments I and evaluate $\varphi[I]$. Are there more efficient algorithms?

It turns out that simply finding out whether an efficient algorithm exists for satisfiability is a famous open problem[2] (with a prize money of a million dollars set by the Clay Mathematics Institute). The good news is that once a satisfying assignment I is found for φ, everyone can check efficiently that $\varphi[I] = \mathsf{T}$. Unsatisfiability, on the other hand, does not have this property: even after taking the time to verify that $\varphi[I] = \mathsf{F}$ for all possible truth assignments I, it appears hard to convince anyone else of this fact (e.g., how do you convince someone that you do not have a brother?).[3] Finally, checking whether a formula is valid is as hard as checking unsatisfiability.

[2] In complexity jargon, checking if a formula is satisfiable is "NP-complete", and finding an efficient algorithm to determine satisfiability would show that P=NP.

[3] In complexity jargon, the unsatisfiability problem is co-NP complete. The major open problem here is whether or not NP=coNP; that is, whether there exists an efficient way of convincing someone that a formula is unsatisfiable.

Claim 6.4. *A formula φ is valid if and only if $\neg\varphi$ is unsatisfiable.*

Proof. The claim essentially follows from definition. If φ is valid, then $\varphi[I] = \mathsf{T}$ for every interpretation I. This means $(\neg\varphi)[I] = \mathsf{F}$ for every interpretation I, and so $\neg\varphi$ is unsatisfiable. The other direction follows similarly. ■

6.2 Logical Inference

Now that we have established the language and semantics (meaning) of propositional logic, let us now reason with it. Suppose we know two facts. First,

"Bob carries an umbrella if it is cloudy and the forecast calls for rain."

Next we know that

"It is not cloudy."

Can we conclude that Bob is not carrying an umbrella? The answer is no. Bob may always carry an umbrella around to feel secure (say in Ithaca).

To make sure that we make correct logical deductions in more complex settings, let us cast the example in the language of propositional logic. Let P be the atom "it is cloudy", Q be the atom "the forecast calls for rain", and R be the atom "Bob carries an umbrella". Then we are given two premises:

$$(P \wedge Q) \to R, \qquad \neg P$$

Can we make the conclusion that $\neg R$ is true? The answer is no, because the truth assignment $P = Q = \mathsf{F}, R = \mathsf{T}$ satisfies the premises, but does not satisfy the conclusion.[4] The next definition formalizes proper logical deductions.

Definition 6.5. A set of formulas $\{\psi_1, \ldots, \psi_n\}$ **entails** a formula ψ, denoted by $\psi_1, \ldots, \psi_n \models \psi$, if for every truth assignment I that satisfies all of $\psi_1, \ldots, \psi_n$, I satisfies ψ.

When $\{\psi_1, \ldots, \psi_n\}$ entails ψ, we consider ψ as a logical consequence of $\{\psi_1, \ldots, \psi_n\}$.

Theorem 6.6. *$\psi_1, \ldots, \psi_n$ entails ψ if and only if $(\psi_1 \wedge \cdots \wedge \psi_n) \to \psi$ is valid.*

Proof. **Only if direction.** Suppose $\psi_1, \ldots, \psi_n$ entails ψ. To show that $\varphi = (\psi_1 \wedge \cdots \wedge \psi_n) \to \psi$ is valid, we need to show that for every truth assignment I, $\varphi[I] = \mathsf{T}$. Consider any truth assignment I; we have two cases:

- $(\psi_1 \wedge \cdots \wedge \psi_n)[I] = \mathsf{F}$. In this case $\varphi[I] = \mathsf{T}$ by definition of implication $(\to)$.
- $(\psi_1 \wedge \cdots \wedge \psi_n)[I] = \mathsf{T}$. Because $\psi_1, \ldots, \psi_n$ entails ψ, we also have $\psi[I] = \mathsf{T}$. This in turn makes $\varphi[I] = \mathsf{T}$.

[4] If we change the first premise to $(P \wedge Q) \leftrightarrow R$, i.e., "Bob carries an umbrella if and only if it is cloudy and the forecast calls for rain", then $\neg R$ is a valid conclusion.

If direction. Suppose $\varphi = (\psi_1 \wedge \cdots \wedge \psi_n) \rightarrow \psi$ is valid. For any truth assignment I that satisfies all of $\psi_1, \ldots, \psi_n$, we have $(\psi_1 \wedge \cdots \wedge \psi_n)[I] = \mathsf{T}$. We also have $\varphi[I] = ((\psi_1 \wedge \cdots \wedge \psi_n) \rightarrow \psi)[I] = \mathsf{T}$ due to validity. Together this means $\psi[I]$ must be true, by observing the truth table for implication ($\rightarrow$). This shows that $\psi_1, \ldots, \psi_n$ entails ψ. ■

Theorem 6.6 gives us further evidence that we have defined implication ($\rightarrow$) correctly. We allow arguments to be valid even if the premise are false.

Axiom Systems

Checking the validity of a formula is difficult (as we discussed, it has been a long standing open question). On the other hand, we perform logic reasoning everyday, in mathematical proofs and in English. An *axiom system* formalizes the reasoning tools we use in a syntactic way (i.e., pattern matching and string manipulations of formulas) so that we may study and eventually automate the reasoning process.

Definition 6.7. An **axiom system** H consists of a set of formulas, called the axioms, and a set of rules of inference. A rule of inference is a way of producing a new formula (think of it as new logical conclusions), given several established formulas (think of it as known facts). A rule of inference has the form:

$$\frac{\begin{array}{c}\varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n\end{array}}{\psi}$$

This means "from the formulas $\varphi_1, \ldots, \varphi_n$ we may infer ψ". We also use the notation $\varphi_1, \ldots \varphi_n \vdash \psi$ (note that this is different from the symbol for satisfiability $\models$).

When we define an axiom system, think of the axioms as as an initial set of tautologies (preferably a small set) that describes our world (e.g., Euclidean geometry has the axiom that two distinct points defines a unique straight line). We can then pattern match the axioms against the rules of inference to derive new tautologies (logical consequences) from the initial set:

Definition 6.8. A **proof** or a **derivation** in an axiom system H is a sequence of formulas $\chi_1, \chi_2, \ldots, \chi_n$ where each formula χ_k either matches[5]an axiom in H, or follows from previous formulas via an inference rule from H, i.e., there exists an inference rule $\varphi_1, \ldots, \varphi_m \vdash \psi$ such that χ_k matches ψ, and there exists $j_1, \ldots, j_m \in \{1, \ldots, k-1\}$ such that χ_{j_i} matches φ_i, correspondingly.

Definition 6.9. We say a formula χ can be inferred from an axiom system H, denoted by $\vdash_H \chi$, if there exists a derivation in H that ends with the formula χ (when understood, we leave out the subscript for convenience).

[5]We have left out the (rather tedious) formal definitions of "matching" against axioms or inference rules. This is best explained through examples later in the section.

Similarly, we say a set of formulas $\varphi_1, \ldots, \varphi_n$ infers χ (under axiom system H), denoted by $\varphi_1, \ldots, \varphi_n \vdash \chi$, if there exists a derivation in H that ends in χ, when $\varphi_1, \ldots, \varphi_n$ are treated as addition axioms.

It is very important to understand that a-priori, derivation and inference has nothing to do with truth and validity. If we start with false axioms or illogical rules of inference, we may end up deriving an invalid formula. On the other hand, if we start with an incomplete set of axioms, or if we miss a few rules of inference, we may not be able to derive some valid formulas. What we want is an axiom system that is both *complete* and *sound*:

Completeness: An axiom system is complete if all valid statements can be derived.

Soundness: An axiom system is sound if only valid statements can be derived.

For example, an axiom system that contains an invalid axiom is not sound, while a trivial axiom system that contains no axioms or no rules of inference is trivially incomplete.

Rules of inference. Here are well-known (and sound) rules of inference for propositional logic:

Modus Ponens:

$$\frac{\varphi \to \psi \quad \varphi}{\psi}$$

Modus Tollens:

$$\frac{\varphi \to \psi \quad \neg\psi}{\neg\varphi}$$

Hypothetical Syllogism:

$$\frac{\varphi \to \psi \quad \psi \to \chi}{\varphi \to \chi}$$

Disjunctive Syllogism:

$$\frac{\varphi \vee \psi \quad \neg\varphi}{\psi}$$

It is easy to see that all of the above inferences rules preserves validity, i.e., the antecedents (premises) entail the conclusion. Therefore an axiom system using these rules will at least be sound.

Example 6.10. The following derivation shows that $\neg C, \neg C \to (A \to C) \vdash \neg A$.

$\neg C$	an axiom
$\neg C \to (A \to C)$	an axiom
$A \to C$	Modus Ponens, from line 1 and 2
$\neg A$	Modus Tollens, from line 3 and 1

In our application of Modus Ponens, we have "matched" $\neg C$ with φ and $(A \to C)$ with ψ.

Example 6.11. The following derivation shows that $A \vee B, \neg B \vee (C \wedge \neg C), \neg A \vdash C \wedge \neg C$. Note that the conclusion is non-sense (can never be true); this is because we have started with a "bad" set of axioms.

$A \vee B$	an axiom
$\neg A$	an axiom
B	Disjunctive Syllogism, from line 1 and 2
$\neg B \vee (C \wedge \neg C)$	an axiom
$C \wedge \neg C$	Disjunctive Syllogism, from line 3 and 4

Axioms. An example axiom may look like this:

$$\varphi \to (\psi \to \varphi)$$

By this we mean any formula that "matches" against the axiom is assumed to be true. For example, let P and Q be atoms, then

$$P \to (Q \to P)$$

$$(P \to Q) \to ((Q \to P) \to (P \to Q))$$

are both assumed to be true (in the second example, we substitute $\varphi = P \to Q$, $\psi = Q \to P$). To have a sound axiom system, we much start with axioms that are valid (tautologies); it is not hard to see that the example axiom is indeed valid.

A sound and complete axiomatization. We now present a sound and complete axiom system for propositional logic. We limit the connectives in the language to only implication ($\to$) and negation ($\neg$); all other connectives that we have introduced can be re-written using only $\to$ and $\neg$ (e.g., $P \vee Q \equiv \neg P \to Q$, $P \wedge Q \equiv \neg(P \to \neg Q)$).

Consider the axioms

$$\varphi \to (\psi \to \varphi) \tag{A1}$$

$$(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)) \tag{A2}$$

$$(\neg\varphi \to \neg\psi) \to (\psi \to \varphi) \tag{A3}$$

Theorem 6.12. *The axioms* (A1)*,* (A2) *and* (A3)*, together with the inference rule Modus Ponens, form a sound and complete axiom system for propositional logic (restricted to connectives $\to$ and $\neg$).*

The proof of Theorem 6.12 is out of the scope of this course (although keep in mind that soundness follows from the fact that our axioms are tautologies and Modus Ponens preserves validity). We remark that the derivations guaranteed by Theorem 6.12 (for valid formulas) are by and large so long and tedious that they are more suited to be generated and checked by computers.

Natural deduction. *Natural deduction* is another logical proof system that generates proofs that appears closer to "natural" mathematical reasoning. This is done by having a large set of inference rules to encompass all kinds of reasoning steps that are seem in everyday mathematical proofs. We do not formally define natural deduction here; instead, we simply list some example rules of inference to present a taste of natural deduction. Note that these are all valid inference rules and can be incorporated into axiom systems as well.

Constructive Dilemma:

$$\frac{\varphi_1 \to \psi_1 \quad \varphi_2 \to \psi_2 \quad \varphi_1 \vee \varphi_2}{\psi_1 \vee \psi_2}$$

Resolution:

$$\frac{\varphi \vee \psi \quad \neg\varphi \vee \chi}{\psi \vee \chi}$$

Conjunction:

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi}$$

Simplification:

$$\frac{\varphi \wedge \psi}{\varphi}$$

Addition:

$$\frac{\varphi}{\varphi \vee \psi}$$

Most of the time we also add rules of "replacement" which allow us to rewrite formulas into equivalent (and simpler) forms, e.g.,

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \qquad \text{De Morgan's Law}$$
$$\neg\neg\varphi \equiv \varphi \qquad \text{Double negation}$$

6.3 First Order Logic

First order logic is an extension of propositional logic. First order logic operates over a set of objects (e.g., real numbers, people, etc.). It allows us to express properties of individual objects, to define relationships between objects, and, most important of all, to quantify over the entire set of objects. Below is a classic argument in first order logic:

All men are mortal
Socrates is a man
Therefore Socrates is a mortal

In first order logic, the argument might be translated as follows:

$$\frac{\forall x\, \text{Man}(x) \to \text{Mortal}(x) \quad \text{Man}(\text{Socrates})}{\text{Mortal}(\text{Socrates})}$$

Several syntax features of first order logic can be seen above: $\forall$ is one of the two quantifiers introduced in first order logic; x is a variable; Socrates is a constant (a particular person); $\text{Mortal}(x)$ and $\text{Man}(x)$ are predicates.

Formally, an **atomic expression** is a *predicate symbol* (e.g., $\text{Man}(x)$, $\text{LessThan}(x, y)$) with the appropriate number of arguments; the arguments can either be *constants* (e.g., the number 0, Socrates) or *variables* (e.g., x, y and z). A first order **formula**, similar to propositional logic, is multiple atomic expressions connected by connectives. The formal recursive definition goes as follows:

- Every atomic expression is a formula.
- If φ and ψ are formulas, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \to \psi$ and $\varphi \leftrightarrow \psi$ are also formulas.
- [New to first order logic.] If φ is a formula and x is a variable, then $\forall x\varphi$ (for all x the formula φ holds) and $\exists x\varphi$ (for some x the formula φ holds) are also formulas.

Example 6.13. The following formula says that the binary predicate P is transitive:

$$\forall x \forall y \forall z (P(x, y) \wedge P(y, z)) \to P(x, z)$$

Example 6.14. The following formula shows that the constant "1" is a multiplicative identity (the ternary predicate $\text{Mult}(x, y, z)$ is defined to be true if $xy = z$):

$$\forall x \forall y (\text{Mult}(1, x, x) \wedge \text{Mult}(x, 1, x))$$

Can you extend the formula to enforce that "1" is the *unique* multiplicative identity?

Example 6.15. The following formula shows that every number except 0 has a multiplicative inverse:

$$\forall x \exists y (\neg \text{Equals}(x, 0) \to \text{Mult}(x, y, 1))$$

Semantics of First Order Logic

We have already described the intuitive meaning of first order logic formulas in English, but let us now give it a formal treatment. Just as in propositional logic, we need an *interpretation* I to assign values to constants, predicates, etc. Additionally, we need a domain D that specifies the universe of objects, in order for quantifiers to make sense.

First we define the notion of a sentence; these are formulas without "dangling" variables.

Definition 6.16. An occurrence of variable x in a formula φ is **bound** if there is some quantifier that operates on x (that is it occurs in some sub-formula ψ the is preceded by $\forall x$ or $\exists x$); otherwise the variable x is **free**. A **sentence** is a formula with no free variables.

Example 6.17. In the following formula (that is not a sentence), the first occurrence of x is free, and the second one is bound:

$$\forall y P(x, y) \to \exists x R(x)$$

The next formula is a sentence (note that in this case, $\exists x$ captures both occurrences of x):

$$\forall y \exists x (P(x, y) \to R(x))$$

From now on we restrict ourselves to sentences, and define their truth values. Recall that in propositional logic, we needed a truth assignment for each propositional atom. In first order logic, we need:

- A **domain** D (simply a set of elements that we are concerned with).
- An **interpretation** $I = I_D$ for domain D that
  - for each constant c, the interpretation assigns an element in the domain $c[I] \in D$.
  - for each predicate $P(x_1, \ldots, x_n)$, the interpretation assigns a function $P[I] : D^n \to \{\mathsf{T}, \mathsf{F}\}$ (equivalently, the interpretation assigns an n-ary relation that contains all n-tuples that evaluates to true).

For example, in the Socrates example, we could have D be the set of all people (or the set of all living creatures, or the set of all Greeks). An interpretation I would need to single out Socrates in D, and also specify for each $a \in D$, whether $\text{Man}(x)$ and $\text{Mortal}(x)$ holds.

Given a first-order sentence φ, a domain D and an interpretation $I = I_D$ (together (D, I) is called a model), we can define the truth value of φ, denoted by $\varphi[I]$, recursively:

- If φ is a atomic expression (i.e., a predicate), then because φ is a sentence, it is of the form $P(c_1, \ldots, c_n)$ where c_i are constants. The value of $\varphi[I]$ is $P[I](c_1[I], \ldots, c_n[I])$.
- If φ has the form $\neg\psi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \to \psi_2$ or $\psi_1 \leftrightarrow \psi_2$, then $\varphi[I] = \neg\psi[I]$, $\psi_1[I] \wedge \psi_2[I]$, $\psi_1[I] \vee \psi_2[I]$, $\psi_1[I] \to \psi_2[I]$ or $\psi_1[I] \leftrightarrow \psi_2[I]$, respectively (following the truth tables for $\neg$, $\wedge$, $\vee$, $\to$, and $\leftrightarrow$).
- If φ has the form $\forall x \psi$, then $\varphi[I]$ is true if and only if for every element $a \in D$, ψ, with free occurrences of x replaced by a, evaluates to true.
- If φ has the form $\exists x \psi$, then $\varphi[I]$ is true if and only if there exists some element $a \in D$ such that ψ, with free occurrences of x replaced by a, evaluates to true.

For instance, if the domain D is the natural numbers $\mathbb{N}$, then

$$\forall x P(x) \equiv P(0) \wedge P(1) \wedge \ldots$$
$$\exists x P(x) \equiv P(0) \vee P(1) \vee \ldots$$

A note on the truth value of first order formulas [optional]. We have cheated in our definition above, in the case that $\varphi = \forall x\psi$ or $\exists x\psi$. When we replace free occurrences of x in ψ by a, we no longer have a formula (because strictly speaking, "a", an element, is not part of the language). One work around is to extend the language with constants for each element in the domain (this has to be done after the domain D is fixed). A more common approach (but slightly more complicated) is to define truth values for all formulas, including those that are not sentences. In this case, the interpretation also needs to assign an element $a \in D$ to each (free) variable x occurring in φ; this is out of the scope of this course.

Satisfiability and Validity

We define satisfiability and validity of first order sentences similar to the way they are defined for propositional logic.

Definition 6.18. Given a domain D and an interpretation I over D, we say (I, D) **satisfies** a formula φ if $\varphi[I] = \mathsf{T}$; in this case we write $D, I \models \varphi$. A formula is φ is **satisfiable** if there exists D and I such that $D, I \models \varphi$, and is unsatisfiable otherwise. A formula is φ is **valid** (of a tautology) if for every D and I, $D, I \models \varphi$,

Logical Reasoning and Axiom Systems

We can defined entailment in the same way it was defined for propositional logic. Just as for propositional logic we can find a complete and sound axiomatization for first-order logic, but the axiom system is much more complex to describe and is out of the scope of this course.

6.4 Applications

Logic has a wide range of applications in computer science, including program verification for correctness, process verification for security policies, information access control, formal proofs of cryptographic protocols, etc.

In a typical application, we start by specifying of "model", a desired property in logic, e.g., we want to check that a piece of code does not create deadlocks. We next describe the "system" in logic, e.g., the piece of code, and the logic behind code execution. It then remains to show that our system satisfies our desired model, using tools in logic; this process is called model checking. Recently Edmund Clark received the Turing award in 2007 for his work on hardware verification using model checking. He graduate with his Ph.D. from Cornell in 1976 with Bob Constable as advisor.

Chapter 7

Graphs

"One graph is worth a thousand logs."
– Michal Aharon, Gilad Barash, Ira Cohen and Eli Mordechai.

Graphs are simple but extremely useful mathematical objects; they are ubiquitous in practical applications of computer science. For example:

- In a computer network, we can model how the computers are connected to each other as a graph. The nodes are the individual computers and the edges are the network connections. This graph can then be used, for example, to route messages as quickly as possible.

- In a digitalized map, nodes are intersections (or cities), and edges are roads (or highways). We may have directed edges to capture one-way streets, and weighted edges to capture distance. This graph is then used for generation directions (e.g., in GPS units).

- On the internet, nodes are web pages, and (directed) edges are links from one web page to another. This graph can be used to rank the importance of each web page for search results (e.g., the importance of a web page can be determined by how many other web pages are pointing to it, and recursively how important those web pages are).

- In a social network, nodes are people, and edges are friendships. Understanding social networks is a very hot topic of research. For example, how does a network achieve "six-degrees of separation", where everyone is approximately 6 friendships away from anyway else? Also known as the small world phenomena, Watts and Strogatz (from Cornell) published the first models of social graphs that have this property, in 1998.

  In Milgram's small world experiment, random people in Omaha, Nebraska were tasked to route a letter to "Mr. Jacobs" in Boston, Massachusetts by passing the letter only to someone they know on a first-name basis. The average number of times that a letter switched hands before reaching Mr. Jacobs was approximately 6! This meant that not

only are people well connected, they can route messages efficiently given only the information of their friends (i.e., knowledge only of a small, local part of the graph). Jon Kleinberg (also from Cornell) gave the first models for social graphs that allow such efficient, local routing algorithms.

Definition 7.1. A **directed graph** G is a pair (V, E) where V is a set of vertices (or nodes), and $E \subseteq V \times V$ is a set of edges. An **undirected graph** additionally has the property that $(u, v) \in E$ if and only if $(v, u) \in E$.

In directed graphs, edge (u, v) (starting from node u, ending at node v) is different from edge (v, u). We also allow "self-loops", i.e., edges of the form (v, v) (say, a web page may link to itself). In undirected graphs, because edge (u, v) and (v, u) must both be present or missing, we often treat a non-self-loop edge as an unordered set of two nodes (e.g., $\{u, v\}$).

A common extension is a *weighted graph*, where each edge additionally carries a weight (a real number). The weight can have a variety of meanings in practice: distance, importance and capacity, to name a few.

Graph Representations

The way a graph is represented by a computer can affect the efficiency of various graph algorithms. Since graph algorithms are not a focus of this course, we instead examine the space efficiency of the different common representations. Given a graph $G = (V, E)$:

Adjacency Matrix. We may number the vertices v_1 to v_n, and represent the edges in a n by n matrix A. Row i and column j of the matrix, a_{ij}, is 1 if and only if there is an edge from v_i to v_j. If the graph is undirected, then $a_{ij} = a_{ji}$ and the matrix A is symmetric about the diagonal; in this case we can just store the upper right triangle of the matrix.

Adjacency Lists. We may represent a graph by listing the vertices in V, and for each vertex v, listing the edges that originates from V (i.e., the set $E_v = \{u \mid (v, u) \in E\}$).

Edge Lists. We may simply have a list of all the edges in E, which implicitly defines a set of "interesting" vertices (vertices that have at least one edge entering or leaving).

If the graph is dense (i.e., has lots of edges), then consider the adjacency matrix representation. The matrix requires storing $O(n^2)$ entries, which is comparable to the space required by adjacency lists or edge lists if the graph is dense. In return, the matrix allows very efficient lookups of whether an edge (u, v) exists (by comparison, if adjacency lists are used, we would need to traverse the whole adjacency list for the vertex u). For sparse graphs, using adjacency lists or edge lists can result in large savings in the size of the representation.[1]

[1] Since the advent of the internet, we now have graphs of unprecedented sizes (e.g., the graph of social networks such as Facebook, or the graph of web pages). Storing and working

Vertex Degree

The degree of a vertex corresponds to the number of edges coming out or going into a vertex. This is defined slightly differently for directed and undirected graphs.

Definition 7.2. In a directed graph $G = (V, E)$, the **in-degree** of a vertex $v \in V$ is the number of edges coming in to it (i.e., of the form $(u, v), u \in V$); the **out-degree** is the number of edges going out of it (i.e., of the form $(v, u), u \in V$). The **degree** of v is the sum of the in-degree and the out-degree.

In an undirected graph the degree of $v \in V$ is the number of edges going out of the vertex (i.e., of the form $(v, u), u \in V$), with the exception that self loops (i.e., the edge (v, v)) is counted twice.

We denote the degree of vertex $v \in V$ by $\deg(v)$.

This seemingly cumbersome definition actually makes a lot of sense pictorially: the degree of a vertex corresponds to the number of "lines" connected to the vertex (and hence self loops in undirected graphs are counted twice). The definition also leads to the following theorem:

Theorem 7.3. *Given a (directed or undirected) graph $G = (V, E)$, $2|E| = \sum_{v \in V} \deg(v)$.*

Proof. In a directed graph, each edge contributes once to the in-degree of some vertex and the out-degree of some, possibly the same, vertex. In an undirected graph, each non-looping edge contributes once to the degree of exactly two vertices, and each self-loop contributes twice to the degree of one vertex. In both cases we conclude that $2|E| = \sum_{v \in V} \deg(v)$. ■

A useful corollary is the "hand shaking lemma":[2]

Corollary 7.4. *In a graph, the number of vertices with an odd degree is even.*

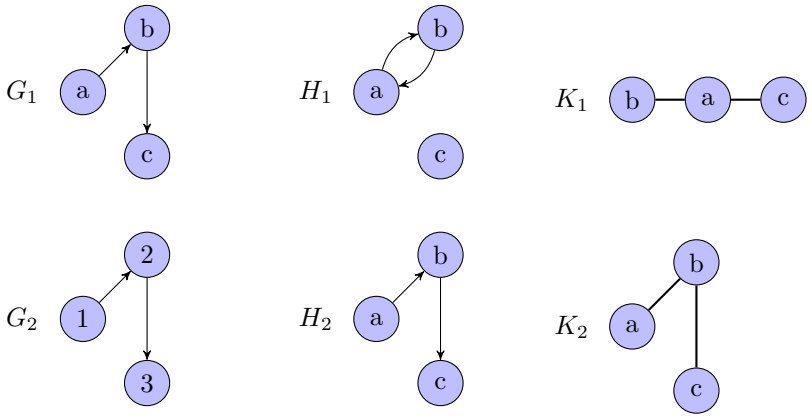
Proof. Let A be the set of vertices of even degree, and $B = V \setminus A$ be the set of vertices of odd degree. Then by Theorem 7.3,

$$2|E| = \sum_{v \in A} \deg(v) + \sum_{v \in B} \deg(v)$$

Since the LHS and the first term of RHS is even, we have that $\sum_{v \in B} \deg(v)$ is even. In order for a sum of odd numbers to be even, there must be a even number of terms. ■

with these graphs are an entirely different science and a hot topic of research backed by both academic and commercial interests.

[2]The name stems from anecdote that the number of people that shake hands with an odd number of people is even.



(a) G_1 and G_2 are clearly isomorphic.

(b) H_1 and H_2 are clearly not isomorphic.

(c) Are K_1 and K_2 isomorphic?

Figure 7.1: Various examples of graph (non-)isomorphism.

7.1 Graph Isomorphism

When do we consider two graphs "the same"? If the number of vertices or edges differ, then clearly the graphs are different. Therefore let us focus on the case when two graphs have the same number of vertices and edges. Consider:

$$G_1 = (V_1 = \{a, b, c\}, E_1 = \{(a, b), (b, c)\})$$
$$G_2 = (V_2 = \{1, 2, 3\}, E_2 = \{(1, 2), (2, 3)\})$$

The only difference between G_1 and G_2 are the names of the vertices; they are clearly the same graph! On the other hand, the graphs

$$H_1 = (V_1 = \{a, b, c\}, E_1 = \{(a, b), (b, a)\})$$
$$H_2 = (V_2 = \{a, b, c\}, E_2 = \{(a, b), (b, c)\})$$

are clearly different (e.g., in H_1, there is a node without any incoming or outgoing edges.) What about the undirected graphs shown in Figure 7.1c? One can argue that K_1 and K_2 are also the same graph. One way to get K_2 from K_1 is to rename/permute the nodes a, b and c to b, c and a, respectively. (Can you name another renaming scheme?)

Definition 7.5. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. The bijection f is called the **isomorphism** from G_1 to G_2, and we use the notation $G_2 = f(G_1)$.

As we would expect, the definition of isomorphism, through the use of the bijection function f, ensures, at the very least, that the two graphs have

the same number of vertices and edges. Another observation is that given an isomorphism f from G_1 to G_2, then inverse function f' is an isomorphism from G_2 to G_1 (we leave it to the readers to verify the details); this makes sense since we would expect isomorphisms to be symmetric.

Given two graphs, wow do we check if they are isomorphic? This is hard problem where no efficient algorithm is known. However, if an isomorphism f is found, it can be efficiently stored and validated (as a proper isomorphism) by anyone. In other words, f serves as a short and efficient proof that two graphs are isomorphic.

Can we prove that two graphs are not isomorphic in an efficient way? Sure, if the graphs have a different number of vertices or edges. Or we may be able to find some structure present in one graph G_1 that can be checked to not be in the other graph G_2, e.g., G_1 contains a "triangle" (three nodes that are all connected to each other) but G_2 doesn't, or G_1 has a vertex of degree 10 but G_2 doesn't. Unfortunately, no general and efficient method is known for proving that two graphs are not isomorphic. This is analogous to the task of proving satisfiability of a propositional formula: if a formula is satisfiable, we can convince others of this efficiently by presenting the satisfying assignment; convincing others that a formula is unsatisfiable seems hard.

Interactive Proofs

In 1985, Goldwasser, Micali and Rackoff, and independently Babai, found a work around to prove that two graphs are not isomorphic. The magic is to add *interaction* to proofs. Consider a proof system that consists of two players, a prover P and a verifier V, where the players may communicate interactively with each other, instead of the prover writing down a single proof. In general the prover (who comes up with the proof) may not be efficient, but the verifier (who checks the proof) must be. As with any proof system, we desire completeness: on input non-isomorphic graphs, the prover P should be able to convince V of this fact. We also require soundness, but with a slight relaxation: on input isomorphic graphs, no matter what the prover says to V, V should reject with very high probability. We present an interactive proof for graph non-isomorphism in Figure 7.2.

Let us check that the interactive proof in Figure 7.2 is complete and sound.

Completeness: If the graphs are not isomorphic, then H is isomorphic to G_b, but not to the other input graph G_{1-b}. This allows P to determine $b' = b$ every time.

Soundness: If the graphs are isomorphic, then H is isomorphic to both G_0 and G_1. Therefore it is impossible to tell from which input graph is used by V to generate H; the best thing P can do is guess. With probability $1/2$, we have $b' \neq b$ and the verifier rejects.

As of now, given isomorphic graphs, the verifier accepts or rejects with probability $1/2$; this may not fit the description "reject with very high probability".

Input: Graphs $G_0 = (V = \{1, \ldots, n\}, E_0)$ and $G_1 = (V = \{1, \ldots, n\}, E_1)$, allegedly not isomorphic.

Step 1: V picks a random permutation/bijection $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, a random bit $b \in \{0, 1\}$, and send $H = \pi(G_b)$ to P.

Step 2: P checks if H is isomorphic to G_0 or G_1, and send $b' = 0$ or $b' = 1$ to V respectively.

Step 3: V accepts (that the graph are non-isomorphic) if and only if $b' = b$.

Figure 7.2: An interactive protocol for graph non-isomorphism (the verifier should accept when G_1 and G_2 are not isomorphic).

Fortunately, we can amplify this probability by repeating the protocol (say) 100 times, and let the verifier accept if only if $b = b'$ is all 100 repetitions. Then by independence, the verifier would accept in the end with probability at most $1/2^{100}$, and reject with probability at least $1 - 1/2^{100}$. Note that the completeness of the protocol is unchanged even after the repetitions.

7.2 Paths and Cycles

Definition 7.6. A **path** or a **walk** in a graph $G = (V, E)$ is a sequence of vertices $(v_0, v_2, \ldots, v_k)$ such that there exists an edge between any two consecutive vertices, i.e. $(v_i, v_{i+1}) \in E$ for $0 \leq i < k$. A **cycle** is a walk where $k \geq 1$ and $v_0 = v_k$ (i.e., starts and ends at the same vertex). The length of the walk, path or cycle is k (i.e., the number of edges).

A directed graph without cycles is called a DAG (a directed acyclic graph). For undirected graphs, we are more interested in cycles that use each edge at most once (otherwise an edge $\{u, v\}$ would induce the "cycle" (u, v, u)). A undirected graph without cycles that use each edge at most once is called a tree. We make a few easy observations:

- On a directed graph, every walk or path is "reversible" (i.e., $(v_k, \ldots, v_0)$ is also a walk/path); this is not necessarily true on a directed graph.
- We allow walks of length 0 (i.e., no walking is done). However cycles must at least have length 1, and length 1 cycles must be a self loop.
- A walk can always be "trimmed" in such away that every vertex is visited at most once, while keeping the same starting and ending vertices.

Example 7.7. The Bacon number of an actor or actress is the shortest path from the actor or actress to Kevin Bacon on the following graph: the nodes are actors and actresses, and edges connect people who star together in a movie.

(a) A strongly connected graph. The graph would not be strongly connected if any edge was removed.

(b) A weakly (but not strongly) connected graph. The graph would not be weakly connected if any edge was removed.

Figure 7.3: The difference between strong and weak connectivity in a directed graph.

The Erdös number is similarly defined to be the distance of a mathematician to Paul Erdös on the co-authorship graph.

Connectivity

Definition 7.8. An undirected graph is **connected** if there exists a path between any two nodes $u, v \in V$ (note that a graph containing a single node v is considered connected via the length 0 path (v)).

The notion of connectivity on a directed graph is more complicated, because paths are not reversible.

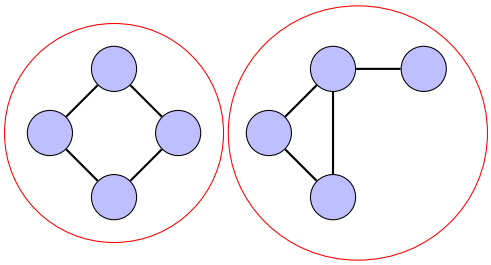
Definition 7.9. A directed graph $G = (V, E)$ is **strongly connected** if there exists a path from any node u to any node v. It is called **weakly connected** if there exists a path from an node u to any node v in the underlying undirected graph: the graph $G' = (V, E')$ where each edge $(u, v) \in E$ in G induces an undirected edge in G' (i.e. $(u, v), (v, u) \in E'$).

When a graph is not connected (or strongly connected), we can decompose the graph into smaller connected components.

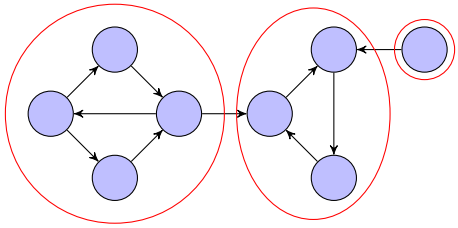
Definition 7.10. Given a graph $G = (V, E)$, a **subgraph** of G is simply a graph $G' = (V', E')$ with $V \subseteq V$ and $E' \subseteq (V' \times V') \cap E$; we denote subgraphs using $G' \subseteq G$.

A **connected component** of graph $G = (V, E)$ is a *maximal* connected subgraph. I.e., it is a subgraph $H \subseteq G$ that is connected, and any larger subgraph H' (satisfying $H' \neq H$, $H \subseteq H' \subseteq G$) must be disconnected.

We may similarly define a **strongly connected component** as a *maximal* strongly connected subgraph.



(a) Connected components of the graph are circled in red. Note that there are no edges between connected components.



(b) *Strongly* connected components of the graph are circled in red. Note that there can still be edges between strongly connected components.

Figure 7.4: Example connected components.

Computing Connected Components

We can visualize a connected component by starting from any node v in the (undirected) graph, and "grow" the component as much as possible by including any other node u that admits a path from v. Thus, first we need an algorithm to check if there is a path from a vertex v to another vertex $u \neq v$.

Breadth first search (BFS) is a basic graph search algorithm that traverses a graph as follows: starting from a vertex v, the algorithm marks v as visited, and traverses the neighbors of v (nodes that share an edge with v). After vising the neighbors of v, the algorithms recursively visits the neighbors of the neighbors, but takes care to ignore nodes that have been visited before. We claim that the algorithm eventually visits vertex u if and only if there is a path from v to u.[3]

To see why, first note that if there are no path between v and u, then of course the search algorithm will never reach u. On the other hand, assume that there exists a path between v and u, but for the sake of contradiction, that the BFS algorithm does not visit u after all the reachable vertices are visited. Let

[3]We have avoided describing implementation details of BFS. It suffice to say that BFS is very efficient, and can be implemented to run in linear time with respect to the size of the graph. Let us also mention here that an alternative graph search algorithm, depth first search (DFS), can also be used here (and is more efficient than BFS at computing strongly connected components in directed graphs).

w be the first node on the path from v to u that is not visited by BFS (such a node must exists because u is not visited). We know $w \neq v$ since v is visited right away. Let w_{-1} be the vertex before w on the path from v to u, which must be visited because w is the *first* unvisited vertex on the path. But this gives a contradiction; after BFS visits w_{-1}, it must also visit w since w is an unvisited neighbor of w_{-1}.[4]

Now let us use BFS to find the connected components of a graph. Simply start BFS from any node v; when the graph search ends, all visited vertex form one connected component. Repeat the BFS on remaining unvisited nodes to recover additional connected components, until all nodes are visited.

Euler and Hamiltonian Cycles

A cycle that uses every edge in a graph exactly once is called a Euler cycle[5] A cycle that uses every vertex in a graph exactly once, except the starting and ending vertex (which is visited exactly twice), is called a Hamiltonian cycle.

How can we find a Euler cycle (or determine that one does not exist)? The following theorem cleanly characterizes when a graph has an Eulerian cycle.

Theorem 7.11. *A undirected graph $G = (V, E)$ has an Euler cycle if and only if G is connected and every $v \in V$ has even degree. Similarly, a directed graph $G = (V, E)$ has an Euler cycle if and only if G is strongly connected and every $v \in V$ has equal in-degree and out-degree.*

Proof. We prove the theorem for the case of undirected graphs; it generalizes easily to directed graphs. First observe that if G has a Euler cycle, then of course G is connected by the cycle. Because every edge is in the cycle, and each time the cycle visits a vertex it must enter and leave, the degree of each vertex is even.

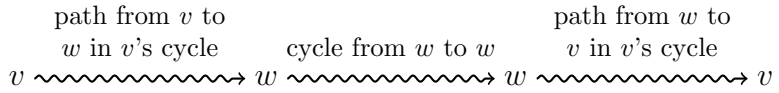
To show the converse, we describe an algorithm that builds the Eulerian cycle assuming connectivity and that each node has even degrees. The algorithm grows the Euler cycle in iterations. Starting from any node v, follow any path in the graph without reusing edges (at each node, pick some unused edge to continue the path). We claim the path must eventually return to v; this is because the path cannot go on forever, and cannot terminate on any other vertex $u \neq v$ do to the even degrees constraint: if there is an available edge into u, there is also an available edge out of u. That is, we now have a cycle (from v to v). If the cycle uses all edges in G then we are done.

Otherwise, find the first node on the cycle, w, that still has an unused edge; w must exist since otherwise the cycle would be disconnected from the part G that still has unused edges. We repeat the algorithm starting from vertex w, resulting in a cycle from w to w that does not have repeated edges, and does

[4] This argument can be extended to show that in fact, BFS would traverse a *shortest* path from v to u.

[5] Originally, Euler was searching for a continuous route that crosses all seven bridges in the city of Königsberg exactly once.

not use edges in the cycle from v to v. We can then "stitch" these two cycles together into a larger cycle:



Eventually the algorithm terminates after finite iterations (since we steadily use up edges in each iteration). When the algorithm terminates, there are no more unused edges, and so we have a Euler cycle. ■

We can relax the notion of Euler cycles into Euler paths — a path that uses every edge in the graph exactly once.

Corollary 7.12. *A undirected graph $G = (V, E)$ has an Euler path, but not a Euler cycle, if and only if the graph is connected and exactly two nodes has an odd degree.*

Proof. Again it is easy to see that if G has a Euler path that is not a cycle, then the graph is connected. Moreover, the starting and ending nodes of the path, and only these two nodes, have an odd degree.

To prove the converse, we *reduce* the problem into finding an Euler cycle. Let $u, v \in V$ be the unique two nodes that have an odd degree. Consider introducing an extra node w and the edges $\{u, w\}, \{v, w\}$. This modified graph satisfies the requirements for having a Euler *cycle*! Once we find the cycle in the modified graph, simply break the cycle at node w to get a Euler path from u to v in the original graph. ■

How can we compute Hamiltonian cycles or paths? Of course we can always do a brute force search on all possible paths and cycles. As of now, no efficient algorithm for computing Hamiltonian cycles (or deciding whether one exists) is known. In fact, this problem is NP-complete, i.e., as hard as deciding whether a propositional formula is satisfiable.

7.3 Graph Coloring

In this section we discuss the problem of coloring the vertices of a graph, so that vertices sharing an edge gets different colors.

Definition 7.13. A (vertex) *coloring* of an undirected graph $G = (V, E)$ is function $c : V \to \mathbb{N}$ (that assigns color $c(v)$ to node $v \in V$) such that nodes that share an edge has different colors, i.e., $\forall (u, v) \in E, c(u) \neq c(v)$.

Definition 7.14. A graph is k-colorable if it can be colored with k colors, i.e., i.e., there exists a coloring c satisfying $\forall v \in V, 0 \leq c(v) < k$. The chromatic number $\chi(G)$ of a graph G is the smallest number such that G is $\chi(G)$-colorable.

Here are some easy observations and special cases of graph coloring:

- A fully connected graph with n nodes (i.e., every two distinct nodes share an edge) has chromatic number n; every node must have a unique color, and every node having a unique color works.
- The chromatic number of a graph is bounded by the max degree of any vertex plus 1 (i.e., $\chi(G) \leq 1 + \max_{v \in V} \deg(v)$). With this many colors, we can color the graph one vertex at a time, by choosing the smallest color that has not been taken by its neighbors.
- A graph is 1-colorable if and only if it does not have edges.
- It is easy to check if a graph is 2-colorable. Simply color an arbitrary vertex v black, then color the neighbors of v white, and the neighbors of neighbors of v black, etc. The graph is 2-colorable if and only if this coloring scheme succeeds (i.e., produces a valid coloring).

In general, determining whether a graph is 3-colorable is NP-complete, i.e., as hard as determining whether a formula is satisfiable.

Coloring Planar Graphs. A graph is planar if all the edges can be drawn on a plane (e.g., a piece of paper) without any edges crossing. A well-known result in mathematics state that all planar graphs are 4-colorable. E.g., the complete graph with 5 nodes cannot be planar, since it requires 5 colors! Also known as the "4 color map theorem", this allow any map to color all the countries (or states, provinces) with only four colors without ambiguity (no neighboring countries will be colored the same). In general, checking whether planar graphs are 3-colorable is still NP-complete.

Applications of Graph Coloring. In task scheduling, if we have a graph whose nodes are tasks, and whose edges connects tasks that are conflicting (e.g., they both require some common resource), then given a coloring, all tasks of the same color can be performed simultaneously. An optimal coloring would partition the tasks into a minimal number of groups (corresponding to the chromatic number).

To decide the number of registers needed to compute a function, consider a graph where the nodes are variables and the edges connect variables that go through a joint operation (i.e., need to be stored on separate registers). Then each color in a coloring corresponds to a need of registers, and the chromatic number is the minimum number of registers required to compute the function.

An Interactive Proof for 3-Coloring

Consider the interactive proof for graph 3-coloring presented in Figure 7.5. Clearly the protocol in Figure 7.5 is complete; if the graph is 3-colorable, the prover will always convince the verifier by following the protocol. What if the graph is not 3-colorable? With what probability can the prover cheat? The coloring $\sigma(c(v))$ must be wrong for at least one edge. Since the verifier V asks

Input: A graph $G = (V, E)$, allegedly 3-colorable.

Step 1: P computes a 3-coloring $c : V \to \{0, 1, 2\}$ of the graph, and picks a random permutation of the three colors: $\sigma : \{0, 1, 2\} \to \{0, 1, 2\}$. P then colors the graph G with the permuted colors (i.e., the coloring $\sigma(c(v))$) and sends the colors to of each vertex to V, but covers each vertex using a "cup".

Step 2: V chooses a random edge $(u, v) \in E$.

Step 3: P removes the cups covering u and v to reveal their colors, $\sigma(c(u))$ and $\sigma(c(v))$.

Step 4: V accepts if and only if $\sigma(c(u)) \neq \sigma(c(v))$.

Figure 7.5: An interactive protocol for graph non-isomorphism (the verifier should accept when G_1 and G_2 are not isomorphic).

the prover P to reveal the colors along a random edge, P will be caught with probability $1/|E|$.

As with before, even though the prover may cheat with a seemingly large probability, $1 - 1/|E|$, we can amplify the probabilities by repeating the protocol (say) $100|E|$ times. Due to independence, the probability that the prover successfully cheats in all $100|E|$ repetitions is bounded by

$$\left(1 - \frac{1}{|E|}\right)^{100|E|} = \left(\left(1 - \frac{1}{|E|}\right)^{|E|}\right)^{100} \approx e^{-100}$$

The zero-knowledge property. It is easy to "prove" that a graph is 3-colorable: simply write down the coloring! Why do we bother with the interactive proof in Figure 7.5? The answer is that it has the *zero-knowledge* property.

Intuitively, in a zero-knowledge interactive proof, the verifier should not learn anything from the interaction other than the fact that the proof statement proved is true. E.g., After the interaction, the verifier cannot better compute a 3-coloring for the graph, or better predict the weather for tomorrow. Zero-knowledge is roughly formalized by requiring that the prover only tells the verifier things that he already knows — that the prover messages could have been generated by the verifier itself.

For our 3-coloring interactive proof, it is zero-knowledge because the prover messages consists only of two random colors (and anyone can pick out two random colors from $\{0, 1, 2\}$).

Implementing electronic "cups". To implement a cup, the prover P can pick an RSA public-key (N, e) and encrypt the color of each node using Padded

RSA. To reveal a cup, the prover simply provide the color and the padding (and the verifier can check the encryption). We use Padded RSA instead of plain RSA because without the padding, the encryption of the same color would always be the same; essentially, the encryptions themselves would give the coloring away.

7.4 Random Graphs [Optional]

A rich subject in graph theory is the study of the properties of randomly generated graphs. We give a simple example in this section.

Consider the following random process to generate a n vertex graph: for each pair of vertices, randomly create an edge between them with independent probability 1/2 (we will not have self loops). What is the probability that two nodes, u, and v, are connected with a path of length at most 2? (This is a simple version of "six degrees of separation".)

Taking any third node w, the probability that the path u–w–v does not exist is 3/4 (by independence). Again by independence, ranging over all possible third nodes w, the probability the path u–w–v does not exist for all $w \neq u, w \neq v$ is $(3/4)^{n-2}$. Therefore, the probability that u and v are more than distance 2 part is at most $(3/4)^{n-2}$.

What if we look at all pairs of nodes? By the union bound (Corollary 5.13), the probably the same pair of node is more than distance 2 apart is

$$\begin{aligned}\Pr\left[\bigcup_{u\neq v} u, v \text{ has distance} \geq 2\right] &\leq \sum_{u\neq v} \Pr[u, v \text{ has distance} \geq 2] \\ &\leq \frac{n(n-1)}{2}\left(\frac{3}{4}\right)^{n-2}\end{aligned}$$

This quantity decreases very quickly as the number of vertices, n increases. Therefore it is most likely that every pair of nodes is at most distance 2 apart.

Chapter 8

Finite Automata

"No finite point has meaning without an infinite reference point."
– Jean-Paul Sartre.

A finite automaton is a mathematical model for a very simple form of computing. As such, we are most interested in what it can and cannot compute. More precisely, a finite automaton takes as input a string of characters (taken from some alphabet Σ, typically $\Sigma = \{0, 1\}$), and outputs "accept" or "reject". Always accepting or rejecting would be considered "easy computing", while accepting if and only if the input string fits a complex pattern is considered "more powerful computing". What power does a finite automaton hold?

8.1 Deterministic Finite Automata

Definition 8.1. A deterministic finite automaton (DFA) is a 5-tuple $M = (S, \Sigma, f, s_0, F)$ where

- S is a finite set of states.
- Σ is a finite input alphabet (e.g 0,1).
- f is a transition function $f : S \times \Sigma \to S$.
- $s_0 \in S$ is the start state (also called the initial state).
- $F \subseteq S$ is a set of final states (also called the accepting states).

Here is how a DFA operates, on input string x. The DFA starts in state s_0 (the start state). It reads the input string x one character at at time, and transition into a new state by applying the transition function f to the current state and the character read. For example, if $x = x_1 x_2 \cdots$, the DFA would start by transitioning through the following states:

$$\underbrace{s^{(0)} = s_0}_{\text{starting state}} \quad \to \quad \underbrace{s^{(1)} = f(s^{(0)}, x_1)}_{\text{after reading } x_1} \quad \to \quad \underbrace{s^{(2)} = f(s^{(1)}, x_2)}_{\text{after reading } x_2}$$

After reading the whole input x, if the DFA ends in an accepting state $\in F$, then x is accepted. Otherwise x is rejected.

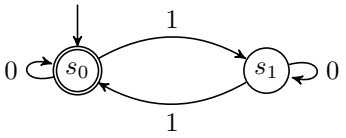
Definition 8.2. Given an alphabet Σ, a **language** L is just a set of strings over the alphabet Σ, i.e., $L \subseteq \Sigma^*$. We say a language L is **accepted** or **recognized** by a DFA M, if M accepts an input string $x \in \Sigma^*$ if and only if $x \in L$.

We frequently illustrate a DFA with a graph: each state $s \in S$ becomes a node, and each mapping $(s, \sigma) \mapsto t$ in the transition function becomes an edge from s to t labeled by the character σ. The start state is usually represented by an extra edge pointing to it (from empty space), while the final states are marked with double circles.

Example 8.3. Consider the alphabet $\Sigma = \{0, 1\}$ and the DFA $M = (S, \Sigma, f, s_0, F)$ defined by

$$S = \{s_0, s_1\} \qquad F = \{s_0\}$$
$$f(s_0, 0) = s_0 \qquad f(s_0, 1) = s_1$$
$$f(s_1, 0) = s_1 \qquad f(s_1, 1) = s_0$$

The DFA M accepts all strings that has an even number of 1s. Intuitively, state s_0 corresponds to "we have seen an even number of 1s", and state s_1 corresponds to "we have seen an odd number of 1s". A graph of M looks like:



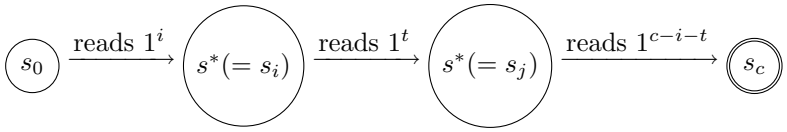
Automata with output. Occasionally we consider DFAs with output. We extend the transition function to have the form $f : S \times \Sigma \to S \times \Sigma$; i.e., each time the automata reads a character and makes a transition, it also outputs a character. Additional extensions may allow the DFA to sometimes output a character and sometimes not.

Limits of Deterministic Finite Automata

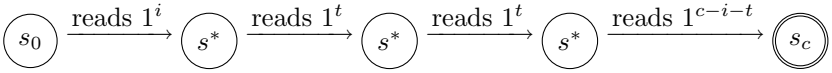
Finite automata can recognize a large set of languages (see regular expressions later), but also have some basic limitations. For example, they are not good at counting.

Lemma 8.4. *Let c be a constant and $L = \{1^c\}$ (the singleton language containing the string of c many 1s). Then no DFA with $< c$ states can accept L.*

Proof. Assuming the contrary that some DFA M with $< c$ states accepts L. Let $s_0, \ldots, s_c$ be the states traversed by M to accept the string 1^c (and so



(a) State transitions of M when it accepts the string 1^c.



(b) State transitions of M when it accepts the string 1^{c+t}.

Figure 8.1: Illustration for Lemma 8.4. If a DFA M with $< c$ states accepts the string 1^c, then it must also accept infinitely many other strings.

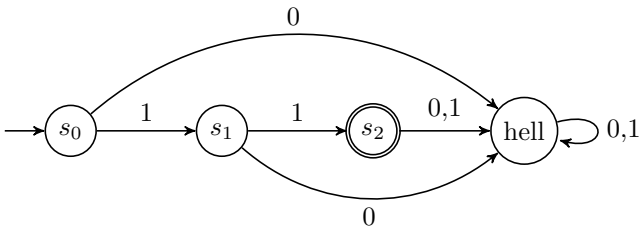


Figure 8.2: A DFA with 4 states that accepts the language $\{1^2\}$. This can be easily generalized to construct a DFA with $c + 2$ states that accepts the language $\{1^c\}$.

$s_c \in F$ is an accept state). By the pigeon hold principle, some state is repeated twice, say $s^* = s_i = s_j$ with $j > i$. Let $t = j - i$.

We now claim that M must also accept the strings 1^{c+t}, 1^{c+2t}, etc; let us describe the behavior of M on these inputs. Starting from s_0, after reading 1^i, M ends up in state $s^* = s_i$. From this point on, for every t many 1s in the input, M will loop back to state $s^* = s_i = s_j$. After sufficiently many loops, M reads the final $c - i - t$ many 1s, and transition from state $s^* = s_j$ to s_c, an accept state. See Figure 8.1.

This gives a contradiction, since M accepts (infinitely) more strings than the language L. ■

On the other hand, see Figure 8.2 for a DFA with $c + 2$ states that accepts the language $\{1^c\}$. The techniques of Lemma 8.1 can be generalized to show the pumping lemma:

Lemma 8.5 (Pumping Lemma)**.** *If M is a DFA with k states and M accepts some string x with $|x| > k$, there there exists strings u, v and w such that $x = uvw$, $|uv| \leq k$, $|v| \geq 1$ and uv^iw is accepted by M for $i \in \mathbb{N}$.*

Proof sketch. Again let $s_0, \ldots, s_{|x|}$ be the states that M travels through to accept the string x¿ Due to the pigeonhole principle, some state must be repeated among $s_0, \ldots, s_k$, say $s^* = s_i = s_j$ with $0 \leq i < j \leq k$. We can now set u to be the first i characters of x, v to be the next $j - i > 0$ characters of x, and w to be the rest of x. ■

Example 8.6. No DFA can accept the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ (intuitively, this is another counting exercise). If we take any DFA with N states, and assume that it accepts the string $0^N 1^N$, then the pumping lemma says that the same DFA must accept the strings $0^{N+t}1^N$, $0^{N+2t}1^N$, etc., for some $0 < t \leq N$.

Example 8.7. No DFA can accept the language $L = \{0^{n^2} \mid n \in \mathbb{N}\}$. If we take any DFA with N states, and assume that it accepts the string 0^{N^2}, then the pumping lemma says that the same DFA must accept the strings 0^{N^2+t}, 0^{N+2t}, etc., for some $0 < t \leq N$. (In particular, $0^{N^2+t} \notin L$ because $0 < t \leq N$.)

A game theory perspective. Having a computing model that does not count may be a good thing. Consider the repeated prisoner's dilemma from game theory. We have two prisoners under suspicion for robbery. Each prisoner may either cooperate (C) or defect (D) (i.e., they may keep their mouths shut, or rat each other out). The utilities of the players (given both players' choices) are as follows (they are symmetric between the players):

	C	D
C	$(3, 3)$	$(-5, 5)$
D	$(5, -5)$	$(-3, -3)$

Roughly the utilities say the following. Both players cooperating is fine (both prisoners get out of jail). But if one prisoner cooperates, the other should defect (not only does the defector get out of jail, he always gets to keep the root all to himself, while his accomplish stays in jail for a long time). If both players defect, then they both stay in jail.

In game theory we look for a stable state called a Nash equilibrium; we look a pair of strategies for the prisoners such that neither player has any incentive to deviate. It is unfortunate (although realistic) that the only Nash equilibrium here is for both prisoners to defect.

Now suppose we repeat this game 100 times. The total utility of a player is $\sum_{i=1}^{100} \delta^i u^{(i)}$ where $u^{(i)}$ the utility of the player in round i, and $0 < \delta < 1$ is a discount factor (for inflation and interests over time, etc.). Instead of prisoners, we now have competing stores on the same street. To cooperate is to continue business as usual, while to defect means to burn the other store down for the day.

Clearly cooperating all the way seems best. But knowing that the first store would cooperate all the time, the second store should defect in that last (100^{th}) round. Knowing this, the first store would defect the round before

(99^{th} round). Continuing this argument[1], the only Nash equilibrium is again for both prisoners to always defect.

What happens in real life? Tit-for-tat[2] seems to be the most popular strategy: cooperate or defect according to the action of the other player in the previous round (e.g., cooperate if the other player cooperated). How can we change our game theoretical model to predict the use of tit-for-tat?

Suppose players use a DFA (with output) to compute their decisions; the input is the decision of the other player in the previous round. Also assume that players need to pay for the number of states in their DFA (intuitively, having many states is cognitively expensive). Then tit-for-tat is a simple DFA with just 1 state s, and the identity transition function: $f(s, C) = (s, C)$, $f(s, D) = (s, D)$. Facing a player that follows tit-for-tat, the best strategy would be to cooperate until round 99 and then defect in round 100. But we have seen that counting with DFA requires many states (and therefore bears a heavy cost)! This is especially true if the game has more rounds, or if the discount factor δ is harsh (i.e., $\ll 1$). If we restrict ourselves to 1-state DFAs, then both players following tit-for-tat is a Nash equilibrium.

8.2 Non-Deterministic Finite Automata

A non-deterministic finite automaton (NFA) is really just a DFA except that for each character in the alphabet, there may be several edges going out of each state. In other words, after reading a character from the input string, an NFA is given a choice to transition to *multiple* states. We formalize this by allowing the transition function to output a set of possible new states.

Definition 8.8. A nondeterministic finite automaton (NFA) is a 5-tuple $M = (S, \Sigma, f, s_0, F)$ where

- S is a finite set of states.
- Σ is a finite input alphabet.
- f is a transition function $f : S \times \Sigma \to \mathcal{P}(S)$.
- $s_0 \in S$ is the start state (also called the initial state).
- $F \subseteq S$ is a set of final states (also called the accepting states).

An NFA M is said to accept an input string x if it is possible to transition from the start state s_0 to some final state $s \in F$. More formally, M is said to accept x if there exists a sequence of states $s^0, s^1, \ldots s^{|x|}$ such that $s^0 = s_0$, $s^{|x|} \in F$ and for each $i \in \{0, \ldots, |x| - 1\}$, $s^{i+1} \in f(s^i, x_i)$. As before, we say M accepts (or recognizes) a language L for all inputs x, M accepts x if and only if $x \in L$. Note that just as it is possible for a state to have multiple

[1] See "Induction and Rationality" in Section 2.5

[2] In a famous example in 2000, Apple removed all promotions of ATI graphic cards after ATI prematurely leaked information on upcoming Macintosh models.

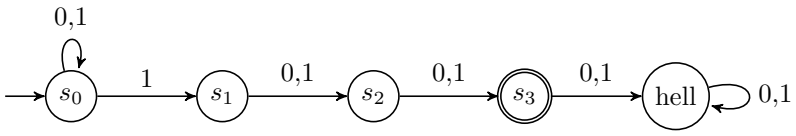


Figure 8.3: A NFA with 5 states that accepts the language L_3. Intuitively, given a string $x \in L_3$, the NFA would choose to remain in state s_0 until it reads the third last character; it would then (magically decide to) transition to state s_1, read the final two characters (transitioning to s_2 and s_3), and accept. The converse that any x accepted by the NFA must be in the language L_3 is easy to see. This can be easily generalized to construct an NFA with $n + 2$ states that accepts the language L_n.

possible transitions after reading a character, a state may have no possible transitions. An input that simply does not have a sequence of valid state transitions (ignoring final states altogether) is of course rejected.

Note that an NFA is not a realistic "physical" model of computation. At any point in the computation where there are multiple possible states to transition into, it is hard to find locally the "correct transition". An alternative model of computation is a *randomized finite automaton* (RFA). A RFA is much like an NFA, with the additional property that whenever there is a choice of transitions, the RFA would specify the probability with which the automaton transitions to each of the allowed states. Correspondingly, a RFA not simply accept or reject an input x, but instead accepts each input with some probability. Compared to an NFA, a RFA is a more realistic "physical" model of computation.

Example 8.9. Consider the language $L_n = \{x \in \{0,1\}^* \mid |x| \geq n, x_{|x|-n} = 1\}$ (i.e., the language of bit strings where the n^{th} bit counting from the end is a 1). L_n can be recognized by a $O(n)$-state NFA, as illustrated in Figure 8.3.

On the other hand, any DFA that recognizes L_n must have at least 2^n states. (Can you construct a DFA for recognizing this language?) Let M be a DFA with less than 2^n states. By the pigeonhole principle, there exists 2 n-bit strings x and x' such that M would reach the same state s after reading x or x' as input (because there are a total of 2^n n-bit strings). Let x and x' differ in position i ($1 \leq i \leq n$), and without loss of generality assume that $x_i = 1$ and $x'_i = 0$. Now consider the strings $\hat{x} = x1^{n-i}$ and $\hat{x'} = x'1^{n-i}$ (i.e., appending the string of $n - i$ many 1s). M would reach the same state after reading $\hat{x}$ or $\hat{x'}$ (since it reached the same state after reading x or x'), and so M must either accept both strings or reject both strings. Yet $\hat{x} \in L_n$ and $\hat{x'} \notin L_n$, i.e., M does not recognize the language L_n.

Clearly, any language recognized by a DFA can be recognized by a NFA since any DFA is a NFA. The following result show that the converse is true too: any NFA can be converted into a DFA that recognizes the same language.

The conversion described below causes an exponential blow-up in the number of states of the automaton; as seen in Example 8.9, such a blow-up is sometimes necessary.

Theorem 8.10. *Let M be an NFA and L be the language recognized by M. Then there exists a DFA M that recognizes the same language L.*

Proof. Let $M = (S, \Sigma, f, s_0, F)$ be an NFA. We construct a DFA $M' = (T, \Sigma, f', t_0, F')$ as follows:

- Let $T = \mathcal{P}(S)$; that is, each state t of M' corresponds to a subset of states of M.
- Upon reading the character $\sigma \in \Sigma$, we transition from state $t \in \mathcal{P}(S)$ to the state corresponding to the *union* of all the possible states that M could have transitioned into, if M is currently in any state $s \in t$. More formally, let
$$f'(t, \sigma) = \bigcup_{s \in t} f(s, \sigma)$$
- The start state of M' is the singleton state containing the start state of M, i.e., $t_0 = \{s_0\}$.
- The final states of M' is any state that contains a final state of M, i.e., $F' = \{t \in T = \mathcal{P}(S) \mid t \cap F \neq \emptyset\}$.

Intuitively, after reading any (partial) string, the DFA M' tries to keep track of all possible states that M may be in.

We now show that the DFA M' accepts any input x if and only if the NFA M accepts x. Assume that M accepts x; that is, there exists some path of computation $s^0, s^1, \ldots, s^{|x|}$ such that $s^0 = s_0$, $s^{|x|} \in F$, and $s^{i+1} \in f(s^i, x_i)$. Consider the (deterministic) path of computation of M' on input x: $t^0, \ldots, t^{|x|}$. It can be shown inductively that $s^i \in t^i$ for all $0 \leq i \leq |x|$:
Base case. $s^0 \in t^0$ since $t^0 = \{s^0\}$ by definition.
Inductive step. If $s^i \in t^i$, then because $s^{i+1} \in f(s^i, x_i)$, we also have

$$s^{i+1} \in t^{i+1} = \bigcup_{s \in t^i} f(s, x_i)$$

We conclude that $s^{|x|} \in t^{|x|}$. Since $s^{|x|} \in F$, we have $t^{|x|} \in F'$ and so M' would accept x.

For the converse direction, assume M' accepts x. Let $t^0, \ldots, t^{|x|}$ be the deterministic computation path of M', with $t^0 = t_0$ and $t^{|x|} \in F'$. From this we can inductively define an accepting sequence of state transitions for M on input x, starting from the final state and working backwards.
Base case. Because $t^{|x|} \in F'$, there exists some $s^{|x|} \in \tau^{|x|}$ such that $s^k \in F$.
Inductive step. Given some $s^{i+1} \in t^{i+1}$, then there must exist some $s^i \in t^i$ such that $s^{i+1} \in f(s^i, x_i)$ (in order for t^i to transition to t^{i+1}.

It is easy to see that the sequence $s^0, \ldots, s^{|x|}$, inductively defined above, is an valid, accepting sequence of state transitions for M on input x: $s^0 = s_0$ since $s^0 \in t^0 = t_0 = \{s_0\}$, $s^{|x|} \in F$ by the base case of the definition, and the transitions are valid by the inductive step of the definition. Therefore M accepts x. ■

8.3 Regular Expressions and Kleene's Theorem

Regular expressions provide an "algebraic" way of specifying a language: namely, we start off with some basic alphabet and build up the language using a fixed set of operations.

Definition 8.11. The set of **regular expressions** over alphabet Σ are defined inductively as follows:

- the symbols "$\boldsymbol{\emptyset}$" and "$\boldsymbol{\varepsilon}$" are regular expressions.
- the symbol "$\mathbf{x}$" is a regular expression if $x \in \Sigma$ (we use boldface to distinguish the symbol $\mathbf{x}$ from the element $x \in \Sigma$).
- if A and B are regular expressions, then so are $A|B$ (their alternation), AB (their concatenation), and A^* (the Kleene star of A).

Usually the Kleene star takes precedence over concatenation, which takes precedence over alternation. In more complex expressions, we use parenthesis to disambiguate the order of operations between concatenations, alternations and Kleene stars. Examples of regular expressions over the lower case letters include: $\mathbf{ab}|\mathbf{c}^*$, $(\mathbf{a}|\mathbf{b})(\mathbf{c}|\boldsymbol{\varepsilon})$, $\boldsymbol{\emptyset}$. A common extension of regular expressions is the "+" operator; $A+$ is interpreted as syntactic sugar (a shortcut) for AA^*.

As of now a regular expression is just a syntactic object — it is just a sequence of symbols. Next we describe how to interpret these symbols to specify a language.

Definition 8.12. Given a regular expression E over alphabet Σ, we inductively define $L(E)$, the **language associated with** E, as follows:

- $L(\boldsymbol{\emptyset}) = \emptyset$ (i.e., the empty set).
- $L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$ (i.e., the set consisting only of the empty string).
- $L(\mathbf{x}) = \{x\}$ (i.e., the singleton set consisting only of the one-character string "x").
- $L(AB) = L(A)L(B) = \{ab \mid a \in L(A), b \in L(B)\}$.
- $L(A|B) = L(A) \cup L(B)$.
- $L(A^*) = L(A)^* = \{\boldsymbol{\varepsilon}\} \cup \{a_1 a_2 \cdots a_n \mid n \in \mathbb{N}^+, a_i \in L(A)\}$. (note that this is a natural extension of the star notation defined in Definition 1.11).

Example 8.13. The parity language consisting of all strings with an even number of 1s can be specified by the regular expression $(\mathbf{0}^*\mathbf{1}\mathbf{0}^*\mathbf{1}\mathbf{0}^*)^*$. The language consisting of all finite strings $\{0,1\}^*$ can be specified either by $(\mathbf{0}|\mathbf{1})^*$ or $(\mathbf{0}^*\mathbf{1}^*)^*$.

Languages that can be expressed as a regular expression are called regular. The class of regular languages are used in a wide variety of applications such as pattern matching or syntax specification.

Definition 8.14. A language L over alphabet Σ is **regular** if there exists a regular expression E (over Σ) such that $L = L(E)$.

Kleene's Theorem

It turns out that DFAs, and thus also NFAs, recognize exactly the class of regular languages.

Theorem 8.15 (Kleene)**.** *A language is regular if and only if it is recognized by a DFA M.*

Proof Sketch. The interesting direction is that any regular language can be recognized by a DFA. Since NFA are equivalent to DFA, it suffices to show that every regular language can be recognized by a NFA. The proof proceeds by induction. It is easy to show that the regular expressions $\boldsymbol{\emptyset}$, $\boldsymbol{\varepsilon}$, and $\mathbf{x}$ for $x \in \Sigma$ can be recognized by a DFA:

- $\emptyset$ is recognized by a DFA without any final states.
- $\boldsymbol{\varepsilon}$ is recognized by a DFA where only the start state is final, and any input leads to a non-final state.
- $\mathbf{x}$ can be recognized by a DFA where the input x takes the initial input s_0 to a final state but no other input does.

Let us show how to recognize $L(AB)$ if the languages $L(A)$ and $L(B)$ can be recognized by NFDAs M_A and M_B.

- Consider a combined machine M_AB where for all the final states in M_A, we add all the connections that the start state in M_B has (i.e., links to states used by M_B). The final states in M_AB are the final states of M_B; furthermore, if the start state of M_B is final, we add back all the final states of M_A (i.e., the final states are now the union of the final states of M_A and M_B).

the same way, we construct machine $M_{A|B}$ and M_{A*} for recognizing the languages $L(A|B)$ and $L(A^*)$.

- $M_{A|B}$ is again a combination of M_A and M_B, but now we instead collapse the start state in M_A and M_B into a single state (keeping all the outgoing links to either M_A or M_B), and let the final set of $M_{A|B}$ be the union of the final sets of M_A and M_B.

- M_{A*} is the machine M_A but where we add all the links going out from the start state to all the final states of M_A.

It is easy to see that the constructed machines recognize the desired language.

It now follows by induction on the number of operators in the regular expression that any languages that can be specified by a regular expression can be recognized by a NFA and thus by a DFA (can you write out the formal inductive proof?). ■