Chapter 6

Additional Set Concepts

Appendix A presents the basics of sets, including notations, operators, and visualizations with Venn diagrams. This chapter assumes that you know that material. If you are feeling a lack of confidence in your knowledge of, for example, set builder notation or the set difference operator, please review that section before continuing.

6.1 What is So Important about Sets?

A good working knowledge of sets is essential in the study of discrete structures. We will define relations, and soon after, functions, in terms of sets. Relational database management systems (RDBMSes) are based on set theory. The concept of finite probability is defined by the ratio of two set cardinalities. If you know data structures such as arrays and linked lists, you likely recognize that they are representations of collections – often sets – of values.

Beyond this class, you may find yourself studying topics in theoretical computer science. Many of those ideas are defined using sets.

Example 100:

Problems in computing that can be solved by performing a quantity of operations that is related polynomially to the size of the problem are known as *polynomial-time* (P) problems. Many sorting algorithms have this property. For example, insertion sort's quantity of operations performed is proportional to n^2 in the worst case, where n is the number of values being sorted.

Problems whose solutions can be verified (just checked for correctness, not necessarily discovered) in polynomial time are known as NP (`non–deterministic polynomial time`) problems. Again using sorting as an example, we can easily check that a list of values is sorted using a quantity of operations that can be described by a linear function. Because linear functions are polynomial, sorting is an NP problem as well as a P problem.

Are all problems in both sets? That is, are the two sets equal? No one knows; it's arguably the most important open question in computer science. Computer scientists and mathematicians are pretty sure these sets are *not* equal; we just haven't been able to prove it (or disprove it).

Why, then, are we pretty sure? Because we know of hundreds of problems for which no efficient solution (that is, no polynomial-time solution) is known. One example is the *partitioning problem*. Consider the set of integers $\{1, 3, 4, 5, 6, 9, 12\}$. Can these values be divided into two subsets such that all of the values are included and the sums of the values in each subset are equal? The answer is 'yes,' both $\{1, 3, 4, 12\}$ and $\{5, 6, 9\}$ have values whose sums are 20. This probably doesn't sound like a very hard problem, and it isn't when the quantity of values is very small, because we can easily enumerate all possible subsets. But imagine that you had to find two such subsets from a set of cardinality 500. The only known algorithm that is guaranteed to find a solution, if one exists, is the algorithm that enumerates all possible subsets. As we will see when we cover counting, the number of possible subset pairs is exponential, not polynomial, in the cardinality of the original set. For that set of cardinality 500, you'd have to examine a ridiculous quantity of subsets.[1] The lack of a better algorithm probably means that this a hard problem.

6.1.1   Sets vs. Logic

Set theory, like proof theory, is frequently considered to be part of mathematical logic because set theory is built on a foundation of logic. As such, much

---

[1]Specifically, $2^{500-1} - 1 = 1,636,695,303,948,070,935,006,594,848,413,799,576,108,$ $321,023,021,532,394,741,645,684,048,066,898,202,337,277,441,635,046,162,952,078,$ $575,443,342,063,780,035,504,608,628,272,942,696,526,664,263,794,687.$ See what happens when you waste time reading footnotes?

of what we will learn about sets has direct parallels with the material we have already covered in Chapter 1. In fact, in this chapter we will spend a fair amount of time explaining how set operators can be defined using logic, and how set identities can be proven using logic.

Thus, it would be incorrect to say that set theory and first-order logic are competitors. They are two systems of expression and reasoning that are closely related. If you are coming into this chapter feeling confident that its content will be easier to follow than the logic chapters because sets already make sense to you, great! Hopefully, the connections we will make between sets and logic will help you understand logic even better, without destroying that warm, fuzzy feeling you have for sets.

6.1.2 Limitations of Sets

Set theory, like logic, doesn't provide the tools necessary to describe everything we might like to describe. This admission shouldn't cause you to lose faith in mathematics in general and computer science in particular. It's clear that the foundations of these fields of study are useful, because computers do work, we are able to launch satellites into orbit, etc. But, just as screwdrivers and hammers are both useful tools for different tasks, set theory is most useful for what it is designed to describe.

Where does set theory start to lose its usefulness? The next example exposes a classic difficulty in basic (a.k.a. fundamental, naive) set theory that is also a good example of a *paradox*, a statement that contradicts itself, usually in a mind-boggling way.

paradox

Example 101:

Consider a men's football team with one player who is a barber. The coach insists that every man be clean-shaven; that is, no player has a moustache or a beard. The barber happily shaves all of the players — but only those players — who do not shave themselves. Who shaves the barber?

That doesn't sound hard to answer: He shaves himself, of course. But consider the conditions of the question again. The barber isn't allowed to shave himself; remember, he shaves the players who do not shave themselves. Thus, the barber must shave himself (he can't go to the barber; he *is* the barber!) but cannot shave himself (the barber shaves only those

---

Figure 6.1: "My brain hurts!" Credit: Monty Python's Flying Circus, Season 3, Episode 6 ("The War Against Pornography"), "Gumby Brain Specialist."

players who do not shave themselves).

This is known in set theory as the Barber Paradox, and is the standard example used to illustrate the more formal Russell's Paradox, raised by logician Bertrand Russell in 1901 as a demonstration of a problem with basic set theory. We know that we can have a set of sets; that is, a set can be a member of a set. Consider the idea of a set that is not a member of itself. Now let's create a collection of such sets: Let S be a set of the sets that are not members of themselves. Is S a member of itself? If it is not (that is, if $S \notin S$), then by its definition S must be $\in S$. But if S is a member of itself ($S \in S$), the definition is violated. The realization that S must both be and not be a member of itself is the paradox. In logic: $S \in S \leftrightarrow S \notin S$, where $S = \{s \mid s \notin s\}$.

How's your brain doing? Happily, there's a solution to Russell's Paradox: Adopt a more restrictive view of sets, such as an 'axiomatic' set theory. Essentially, an *axiomatic set theory* is one that adds a group of axioms ('ground

axiomatic set theory

rules') to the basic set ideas so that problems like Russell's Paradox can no longer arise. For our purposes, knowing that there is a solution to such problems is enough.[2]

6.2 Logical Representation of Basic Set Operators

Expressing set operators using logical notation may seem to be a purely intellectual exercise. After all, we can use the operators just fine given their descriptions in English. Without the logic, though, proving set properties that use those operators requires exhaustive enumerations, much like the truth tables we used to prove logical identities. By expressing set operators in logic, we can employ the logical identities we've already proved (or could prove if necessary), hopefully saving time and effort.

6.2.1 Logical Representations of Union, Intersection, Difference, and Complement

All four of the set operators union ($\cup$), intersection ($\cap$), difference ($-$), and complement ($\overline{\Box}$) accept a set or two as operands and produce a set as their result. To represent them in logic, then, we need a notation that allows us to describe their behavior with logic but produce a set. Fortunately, we already have one: Set-builder notation (see A.3.1).

Let's start with union. In Appendix A, we described the union of two sets as being "a binary operator that combines the elements from the two operand sets into a single result set." Thus, an element is in the result if it is found in the first operand set or the second operand set. Because an element found in both operand sets is included in the result, we need an inclusive OR. That's all we need to know to express union logically:

$$A \cup B = \{c \mid c \in A \lor c \in B\}$$

Intersection can be expressed almost exactly the same way. To be in the result, an element must be in the first operand set and the second:

$$D \cap E = \{f \mid f \in D \land f \in E\}$$

Remembering which logical operator pairs with each of these set operators is easy. Operators that open in the same direction go together: $\cap$ goes with $\land$ (both open to the bottom) and $\cup$ goes with $\lor$ (both open to the top).

Difference requires a little more thought to express in logic. Difference 'keeps' all of the elements of the first operand set, unless the element is also

---

[2]If you would like to read about a specific example of an axiomatic theory of sets, look up "ZFC" (Zermelo-Fraenkel set theory with the Axiom of Choice).

found in the second set. That is, an element is in the result set when it is a member of the first set and not a member of the second:

$$G - H = \{i \mid i \in G \land i \notin H\}$$

Complement is the only unary operator of this group, which makes it the easiest to express in logic. The complement of a set is the set of all of the other elements in the universe; that is, everything not in the operand set:

$$\overline{J} = \{k \mid k \notin J\}$$

At this point, you may be wondering if set membership itself can be – should be! – expressed logically, too. After all, it can easily be viewed as a predicate. In practice, we think of set membership as a concept of sets, rather than an operation on a set. As such, we don't consider it to be an operator, though we use it like one.

6.2.2   A New Set Operator: Cartesian Product

A very useful set operator, Cartesian Product pairs up elements of its operand sets. Before we can define Cartesian Product, we need to understand how to combine set elements into an ordered pair.

ordered pair

Definition 27: Ordered Pair

An *ordered pair* (denoted "(a, b)") is a two-element pairing of set elements in which position matters.

The word "ordered" is important, as Example 102 demonstrates.

Example 102:

Let $S = \{4, 6, 2\}$. $(2, 4)$ is an ordered pair of elements from S.

$(4, 2)$ is a different ordered pair of A's elements. That is, $(2, 4) \neq (4, 2)$. This distinguishes ordered pairs from sets of cardinality two, where order doesn't matter: $\{2, 4\} = \{4, 2\}$.

Now we can define Cartesian Product in terms of ordered pairs.

Definition 28: Cartesian Product   *cartesian product*

The *Cartesian Product* of two sets L and M, denoted $L \times M$ (LaTeX: `\times`), is the set of all possible ordered pairs (l, m) such that $l \in L$ and $m \in M$.

In logic: $L \times M = \{(l, m) \mid l \in L \land m \in M\}$

This definition is limited to just two operand sets. We can generalize the idea to work with more than two operands, and will soon. However, to start, we'll stay within this definition.

Example 103:

Problem: Compute $\{K, Q\} \times \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$.

Solution: We need to pair each element of the first set with each element of the second, in that order, to form all of the ordered pairs of the result set. The result: $\{(K, \clubsuit), (K, \diamondsuit), (K, \heartsuit), (K, \spadesuit), (Q, \clubsuit), (Q, \diamondsuit), (Q, \heartsuit), (Q, \spadesuit)\}$

Note that this shows us a way to define the common French deck of 52 playing cards as a set of ordered pairs: $\{A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2\} \times \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$.

When enumerating the result of a Cartesian Product manually, it is important to be systematic, so that no ordered pairs are missed or are duplicated. Counting the quantity of ordered pairs generated and comparing that sum against the fact that $|L \times M| = |L| \cdot |M|$ can detect simple errors. Continuing Example 103, we can verify that our French deck must consist of $|\{A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2\}| \cdot |\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}| = 13 \cdot 4 = 52$ ordered pairs (a.k.a. cards).

Example 104:

Problem: Compute $\emptyset \times \{1, 2\}$.

Solution: To create an ordered pair, we need one element from each

---

operand set. As the empty set has no elements, there can be no ordered pairs. Thus, $\emptyset \times \{1, 2\} = \emptyset$.

As mentioned above, although we won't have a lot of need for it in this book, computing Cartesian Products of more than two sets, or, equivalently, of sets of ordered pairs, is a straight-forward extension of the two-set definition. The results are sets of n-tuples (ordered lists of n elements each).

Example 105:

Problem: Compute $\{a, b\} \times \{\infty\} \times \{3, 5\}$.

Solution: The result is $\{(a, \infty, 3), (a, \infty, 5), (b, \infty, 3), (b, \infty, 5)\}$. Let's see how to compute this one Cartesian Product at a time.

We work left to right. The result of the first operator is as expected: $\{a, b\} \times \{\infty\} = \{(a, \infty), (b, \infty)\}$.

Before computing the second operator's result, it's helpful to think of $\{3, 5\}$ as being a set of 1-tuples: $\{(3), (5)\}$. That's what it really is; we don't ordinarily include the parentheses because there aren't multiple values for them to group together. But, adding the parentheses makes it easier for us to see that we are creating the Cartesian Product of a set of 2-tuples with a set of 1-tuples, which will form the set of 3-tuples listed above: $\{(a, \infty, 3), (a, \infty, 5), (b, \infty, 3), (b, \infty, 5)\}$.

In general, the idea of Example 105 is called an n-ary Cartesian Product.

---

n-ary cartesian product

Definition 29: n-ary Cartesian Product

The n-ary Cartesian Product of n sets S_1 through S_n, $S_1 \times S_2 \times \ldots \times S_n$, produces the set of n-tuples $\{(s_1, s_2, \ldots, s_n) \mid s_i \in S_i\}$.

The quantity of n-tuples in the resulting set is the product of the cardinalities of the n operand sets: $|S_1 \times S_2 \times \ldots \times S_n| = |S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

You're probably wondering: "Why isn't the result of Example 105 written as $\{((a, \infty), 3), ((a, \infty), 5), ((b, \infty), 3), ((b, \infty), 5)\}$?" That is another way to view things, but it's not how the generalization of Cartesian Product is defined. This is a practical choice: In a later chapter, we'll see that n-tuples (and n-ary Cartesian Products) are used by relational database systems to store (and process) data. This isn't to say that the concept of ordered n-tuples that contain other ordered n-tuples doesn't have uses; it does (for example, see nested lists in the programming language Scheme), but they aren't as common in set theory.

6.3   Some Useful Set Concepts

Appendix A introduced the idea of set membership. Earlier in this chapter we said we'll consider set membership to be a concept rather than an operator, even though it could be viewed as an operator. There are other set concepts that must be understood in order to use sets, and much of the rest of this book, effectively. In this section we introduce them. We won't call any of these concepts operators, either, although cases could be made for many of them.

6.3.1   Subsets

Imagine a class of students. Some of them will end the term with the highest grade allowed by the school. That group may be considered a *subset* of the original set of all of the students in the class, because it contains only students that are members of that set of students. Sounds simple enough, but what if no student earns that highest grade? Is that set (the empty set) considered a subset of the class? And what if all of the students earn that grade? Is the class a subset of itself? Such extreme cases are why there are two definitions of 'subset' for us to learn.

Definition 30: Subset   *subset*

Set N is a *subset* of set O (denoted $N \subseteq O$, LaTeX: `\subseteq`) when every element of N is also an element of O. Equivalently, O is a *superset* of N ($O \supseteq N$, LaTeX: `\supseteq`).

In logic: $N \subseteq O \equiv \forall p \ (p \in N \rightarrow p \in O), p \in \mathcal{U}$

This definition deserves some explanation. First, notice that the logical expression for subset doesn't use set builder notation. This is because the 'result' of subset is a logical value (true or false) instead of a set: Either N is a subset of O or it isn't. This explains why we use $\equiv$ instead of $=$; sets can be equal but logical expressions are equivalent.

Time to start considering those extreme cases. The definition doesn't directly address them, but we can infer how they are handled. Let's start with the situation in which $N = O$. Is $N \subseteq O$? That is, is a set a subset of itself? Clearly, yes. The definition says that N is a subset of O when all of N's content is found within O. This is certainly the case when the sets have the same content.

Now consider the other extreme, when $|N| = 0$ (that is, $N = \emptyset$). Is $\emptyset \subseteq O$ true? Yes! The empty set has no elements. Vacuously, all of its elements are found within O. For this reason, the empty set is considered to be a subset of any set, even itself.

We'll see some examples after we introduce the other subset definition: *proper subset*.

proper subset

> **Definition 31: Proper Subset**
>
> Set Q is a *proper subset* of set R (denoted $Q \subset R$, LaTeX: `\subset`) when $Q \subseteq R$ but $Q \neq R$. Equivalently, R is a *proper superset* of Q ($R \supset Q$, LaTeX: `\supset`).
>
> In logic: $Q \subset R \equiv \forall s\,(s \in Q \rightarrow s \in R) \wedge \exists t(t \notin Q \wedge t \in R), \text{where } s, t \in \mathcal{U}$

(Be aware that some people use the symbol $\subsetneq$ (LaTeX: `amssymb`'s `\subsetneq`) for proper subset.)

The difference in meaning between $\subseteq$ and $\subset$ is simple (a set is a subset of itself but not a proper subset of itself), but expressing that difference logically takes some work. To show that two sets Q and R are not equal, we say that R has an element that Q does not. We say it that way in this definition because $|Q| < |R|$ when $Q \subset R$.

People sometimes have trouble keeping the meanings of $\subseteq$ and $\subset$ straight. This observation usually helps: Think of the symbol $\subseteq$ as the result of the merging of the symbols $\subset$ and $=$. This justifies the inclusion of the single

horizontal line in the symbol for subset, and reminds us that the idea of set equality is included in the definition of subset.

Example 106:

The following table shows the similarities of and differences between the subset operators:

	S	T	$S \subseteq T$?	$S \subset T$?
(a)	$\{1,2\}$	$\{1,2,3\}$	true	true
(b)	$\{4,5\}$	$\{5\}$	false	false
(c)	$\{6\}$	$\{6\}$	true	false
(d)	$\emptyset$	$\{7\}$	true	true
(e)	$\emptyset$	$\emptyset$	true	false
(f)	$\{\emptyset\}$	$\{\emptyset, 8\}$	true	true

The examples of line (d) follow from the fact that the empty set is considered to be a subset (and proper subset) of any non-empty set. Line (e) mirrors line (c); as with any other set, the empty set is a subset but not a proper subset of itself. Line (f) is included to make the point that sets can contain other sets, even the empty set, as elements, and when they do, the subset definitions don't change. That is, in line (f), you could replace the two occurrences of $\emptyset$ with another set, say $\{9\}$, and the results would be the same.

6.3.2 Set Equality

Set equality sounds like a concept that's so basic that it should have been covered back in Appendix A. We saved it for this chapter because its definition can be so easily expressed using the idea of subset.

Definition 32: Set Equality

Sets V and W are equal (denoted $V = W$) iff $V \subseteq W$ and $W \subseteq V$.

set equality

We aren't providing the logical version of equality because it's a straightforward application of the logical construction of subset.

Example 107:

Let $S = \{7, 15, 16\}$ and $T = \{16, 7, 15\}$. Applying the set equality definition, we see that $S = T$ because both $S \subseteq T$ and $T \subseteq S$ are true.

We can reach the same conclusion by inspecting the set elements. Despite the different orderings of the elements, S and T have the same cardinality and contain the same elements. The subset-based definition is more generally useful than is the inspection approach. For example, it will serve as the foundation of set equality proofs by cases in Section 6.4.2.

Example 108:

Problem: Let $A = \emptyset$ and $B = \{\emptyset\}$. Does $A = B$?

Solution: No, because these are different sets with different cardinalities: $|A| = 0$ (the empty set has no elements) and $|B| = 1$ (the empty set is the only element of B).

6.3.3   Power Sets

A companion idea to subset is that of *power set*.

---

power set

Definition 33: Power Set

The *power set* of a set S (denoted $\mathcal{P}(S)$, LATEX: \mathcal{P}), is the set of all of S's subsets.

There are a variety of symbols that are used to represent a power set, but this calligraphic 'P' is the most commonly used today.

Remember that the empty set is a subset of any set, making the empty set an element of any set's power set. Also remember that the definition of power set is based on the definition of subset, not of proper subset, meaning that the set itself is another member of the power set.

Example 109:

Problem: Let $A = \{e, f, g\}$. What is $\mathcal{P}(A)$?

Solution: Computing power sets is one of those activities that attracts silly mistakes like a kitchen floor attracts the buttered side of toast. To reduce the chance of an error, be systematic when listing the subsets: Start with all of the subsets of size 0, then list those of size 1, etc., ending with those of size $|A|$.

There is just one subset of size 0: The empty set. There are $|A| = 3$ subsets of size 1: $\{e\}, \{f\}$, and $\{g\}$. There are also three subsets of size 2: $\{e, f\}, \{e, g\}$, and $\{f, g\}$. Finally, the set itself is the only subset of size 3. Thus, $\mathcal{P}(A) = \{\emptyset, \{e\}, \{f\}, \{g\}, \{e, f\}, \{e, g\}, \{f, g\}, \{e, f, g\}\}$.

A tidbit of useful power set information: $|\mathcal{P}(S)| = 2^{|S|}$. Combined with a systematic listing of the subsets by size, knowing this can help you with power set problems that might seem baffling at first glance.

Example 110:

Problem: Let $E = \{\emptyset, \{\emptyset\}\}$. What is $\mathcal{P}(E)$?

Solution: Let's start by using what we just learned: $|\mathcal{P}(E)| = 2^{|E|} = 2^2 = 4$. Now we just have to identify those four subsets.

Two are easy: All sets have the empty set and itself as subsets. We just need the other two, which each must be of size 1, as we've covered size 0 and size 2. Each of these two subsets must contain one of the individual elements of E. We're done: $\mathcal{P}(E) = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$.

If you're wondering how to prove that $|\mathcal{P}(S)| = 2^{|S|}$, great! See the counting chapter for the proof.

6.3.4 Disjoint Sets

Sets that have no common elements are *disjoint*.

> **Definition 34: Disjoint Sets**
>
> Sets S and T are *disjoint sets* when $S \cap T = \emptyset$.

There is no recognized symbol for disjoint sets.

Example 111:

Problem: Explain why the sets $\{i, k, m, o, q\}$ and $\{r, p, n, l, i\}$ are not disjoint.

Solution: By the definition of disjoint sets, for these sets to be disjoint the intersections of these sets must be the empty set. These sets have the element i in common; that is, $\{i, k, m, o, q\} \cap \{r, p, n, l, i\} = \{i\} \neq \emptyset$. Thus, these sets are not disjoint.

Example 112:

Problem: Consider the set $C = \{a, e\}$. How many subsets of the universe $\mathcal{U} = \{a, e, i, o, u\}$ are disjoint with C?

Solution: We need to figure out how many subsets there are of $\mathcal{U}$ that do not contain a or e. This is the same as asking how many subsets there are of $\{i, o, u\}$. We already know that answer, because that collection of subsets is the power set of $\{i, o, u\}$, and its cardinality is $2^{|\{i,o,u\}|} = 8$. There are 8 subsets of $\{a, e, i, o, u\}$ that are disjoint with C.

Example 112 is a good demonstration of how complex problems can be made much less so by looking at the problem from a different point of view. We could find the answer by listing all of the subsets of a five-element universe (all 32 of them) and counting those that do not contain a or e. A little thought

saved a lot of work. We will see this problem-solving technique frequently in the counting chapter. (And yes, this example is a counting problem!)

The idea of disjoint sets can be easily extended to more than a pair of sets by computing all of the pairwise intersections of the sets and verifying that all of those intersections are empty.

Example 113:

Problem: Are the sets $\{11, 15\}$, $\{19\}$, $\{10, 13, 14, 16\}$, and $\{12, 15, 18\}$ all disjoint from one another?

Solution: No, because there is one pair of sets (the first and the fourth) that share 15.

6.3.5 Partitions

We need to make a brief detour into the ideas of *generalized union* and *generalized intersection*, because we need generalized union to conveniently define the idea of a partition.

generalized union
generalized intersection

We know that the symbol $\cup$ is used to represent the union of two sets. But what if we have a large group of sets and want to represent the unioning of all of them? Instead of creating a massive expression $(A \cup B \cup C \cup \ldots)$, we can create a short-cut in the same way that we use Σ to represent a large summation. That is:

$$\bigcup_{i=1}^{n} S_i = S_1 \cup S_2 \cup \ldots \cup S_n.$$

(LaTeX : `\bigcup\limits`, or just `\bigcup` for a more compact version.) Similarly, we can use $\bigcap$ (LaTeX: `\bigcap`) to represent a lengthy intersection expression. End of detour!

Partitions of a set are closely related to the idea of disjoint sets, as the definition makes clear.

Definition 35: Partition

partition

> Non-empty sets $S_1, S_2, \ldots S_n$ form a *partition* of a base set B when $\bigcup_{i=1}^{n} S_i = B$ and $S_j \cap S_k = \emptyset$ for all j, k such that $1 \leq j, k \leq n$.

That is, a group of sets is a partition of B if (a) each set is non-empty, (b) each element of B is found in one of the sets, and (c) all of the sets of the group are disjoint from one another.

Example 114:

Problem: Together, are the set of odd integers ($\mathbb{Z}^{\text{odd}}$) and the set of even integers ($\mathbb{Z}^{\text{even}}$) a partition of $\mathbb{Z}$?

Solution: Yes! All integers are either odd or even (yes, zero, too; it's even), and no integer is both. Because $\mathbb{Z}^{\text{odd}}$ and $\mathbb{Z}^{\text{even}}$ are not empty, every integer is either $\in \mathbb{Z}^{\text{odd}}$ or $\in \mathbb{Z}^{\text{even}}$ (not both), and $\mathbb{Z}^{\text{odd}} \cup \mathbb{Z}^{\text{even}} = \mathbb{Z}$, $\mathbb{Z}^{\text{odd}}$ and $\mathbb{Z}^{\text{even}}$ form a partition of $\mathbb{Z}$.

Let's revisit Example 113's sets from the perspective of a partition.

Example 115:

Problem: Collectively, are the sets $\{11, 15\}$, $\{19\}$, $\{10, 13, 14, 16\}$, and $\{12, 15, 18\}$ a partition of the set $B = \{i \mid 10 \leq i \leq 19, i \in \mathbb{Z}\}$?

Solution: (This is going to sound familiar ...) No, because there is one pair of sets (the first and the fourth) that share 15. For the group to be a partition, each element must be a member of exactly one set.

Let's drop the second occurrence of 15 and try it again.

Example 116:

Problem: Are the sets $\{11, 15\}$, $\{19\}$, $\{10, 13, 14, 16\}$, and $\{12, 18\}$ a partition of the set $B = \{i \mid 10 \leq i \leq 19, i \in \mathbb{Z}\}$?

> *Solution*: Again, no, but for a different reason. For the collection of sets to be a partition, each value of the base set B must be included in one of the sets. $|B| = 10$ but the sum of the cardinalities of the partition sets is just 9. The element 17 is missing.

6.4   Set Identities

The connection between logic and sets extends to their collections of identities. In this section, we will see that these identities are very similar (which is good, as it means that they are easier to remember and to apply). Just as logical identities (see Section 1.5.2) are useful in proving the equivalence of logical expressions, set identities are useful in proving set equivalences. We can also 'jump' between set and logic notations to prove the set identities, should we feel the need.[3]

6.4.1   Common Set Identities

As we did with the logical identities back in Chapter 1, we have divided the set identities into separate tables by the set operators they use. Our collection of set identities isn't nearly as extensive as is our lists of logical identities. That's not to imply that set identities are less important; rather, we don't have as many operators. We don't have corresponding set operators for exclusive-OR, implication, and biimplication.[4] We do have set difference and Cartesian Product, but the latter makes sets of ordered pairs, not of basic elements, with the result that it doesn't 'mingle' well with the others.

> **Table 16: Some Set Identities using Union ($\cup$) and Intersection ($\cap$)**

---

[3]And we will!

[4]Before those of you with set theory backgrounds — you know who you are — get too worked up about this claim, remember that this chapter didn't cover symmetric difference, and we aren't considering subset and proper subset to be operators. Still need to get lathered up? Please address all complaints to `/dev/null`.

(a)	$S \cap S = S$ $S \cup S = S$	Idempotency
(b)	$S \cap \emptyset = \emptyset$ $S \cup \mathcal{U} = \mathcal{U}$	Domination
(c)	$S \cap \mathcal{U} = S$ $S \cup \emptyset = S$	Identity
(d)	$S \cap T = T \cap S$ $S \cup T = T \cup S$	Commutativity
(e)	$S \cap (T \cap W) = (S \cap T) \cap W$ $S \cup (T \cup W) = (S \cup T) \cup W$	Associativity
(f)	$S \cap (T \cup W) = (S \cap T) \cup (S \cap W)$ $S \cup (T \cap W) = (S \cup T) \cap (S \cup W)$	Distributivity
(g)	$S \cap (S \cup T) = S$ $S \cup (S \cap T) = S$	Absorption Laws

Table 17: Some More Set Identities (adding Complement ($\overline{\square}$))

(a)	$\overline{\overline{S}} = S$	Dbl. Complement (Involution)
(b)	$S \cap \overline{S} = \emptyset$ $S \cup \overline{S} = \mathcal{U}$	Complement Laws
(c)	$\overline{S \cap T} = \overline{S} \cup \overline{T}$ $\overline{S \cup T} = \overline{S} \cap \overline{T}$	De Morgan's Laws

Table 18: Still More Set Identities (adding Difference ($-$))

(a)	$S - T = S \cap \overline{T}$	Definition of Difference
(b)	$S - T = S - (S \cap T)$	
(c)	$S - S = \emptyset$	
(d)	$S - \emptyset = S$	
(e)	$\emptyset - S = \emptyset$	
(f)	$S - \mathcal{U} = \emptyset$	
(g)	$\mathcal{U} - S = \overline{S}$	

6.4.2   Proving Set Expressions

You probably remember that, in Chapter 1, we demonstrated how to prove the truth of logical equivalences. At the time, we didn't know what a 'proof' was,[5] but now we know that what we were doing was constructing straight-forward direct proofs. We will do the same sort of thing to prove set expressions.

Let's ease into these proofs by starting with an easy one. On rare occasions, we get lucky and are asked to prove something that's covered by a definition.

Example 117:

Problem: Prove or disprove: $A \subseteq \mathcal{P}(A)$, where A is a set.

Solution: Of course, we already know that a set's power set contains the set itself, but we can't hold our noses in the air, sniff derisively, and walk stiffly away, in search of a conjecture that is up to our standards. We need to prove it if we can, and we definitely can.

Proof (Direct): By definition, the power set of a set S is the set containing all of S's subsets. Because every set is a subset of itself (according to the definition of subset), each set is an element of its own power set.

Therefore, $A \subseteq \mathcal{P}(A)$.

---

[5]Ah, those happier days of our youth . . .

Figure 6.2: Mark Messier, Hall of Fame hockey player and now namesake of all inelegant proofs. (Credit: NY Daily News)

We hope you enjoyed Example 117; the rest of the examples will be much messier.[6]

Example 118:

Problem: Prove or disprove: If $A \subset B$ and $B \subseteq C$, then $A \subset C$.

Solution: As always, it's a good idea to check that the conjecture is worth trying to prove. A quick Venn diagram (see Figure 6.3) suggests that the conjecture holds.

To create a formal proof, we will make use of our old friend, logic. We

---

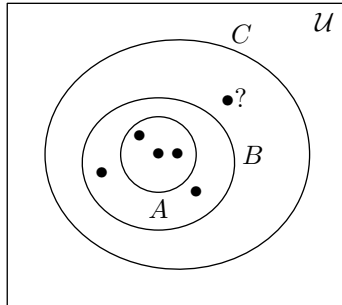[6]Yes, this word is the entire justification for Figure 6.2. We should be ashamed, but aren't.

Figure 6.3: Yes, when $A \subset B$ and $B \subseteq C$, it seems that $A \subset C$ is true. (See Example 118.)

know how to express $\subset$ and $\subseteq$ logically, and we know lots of logical equivalences and several common rules of inference. We'll just put some of them to work. Here's the complete proof; a dissection follows.

Proof (Direct): Assume that the elements represented by d and e, below, are members of the same universe as are the elements of the sets A, B, and C.

(1)	$A \subset B$	[Given]
(2)	$B \subseteq C$	[Given]
(3)	$\forall d(d \in A \to d \in B) \land \exists e(e \notin A \land e \in B)$	[1, Def. of $\subset$]
(4)	$\forall d(d \in B \to d \in C)$	[2, Def. of $\subseteq$]
(5)	$\forall d[(d \in A \to d \in B) \land (d \in B \to d \in C)] \land \exists e(e \notin A \land e \in B)$	
		[3, 4, Conjunction]
(6)	$\forall d[(d \in A \to d \in B) \land (d \in B \to d \in C)] \land \exists e(e \notin A \land e \in C)$	
		[5, Modus Ponens]
(7)	$\forall d(d \in A \to d \in C) \land \exists e(e \notin A \land e \in C)$	[6, Hypo. Syll.]
(8) $\therefore$	$A \subset C$	[7, Def. of $\subset$]

Therefore, if $A \subset B$ and $B \subseteq C$, then $A \subset C$.

We have two useful pieces of given information, making a direct proof seem like a good choice of proof technique. As it does for logical proofs, a tabular format works well for set proofs, although, as you can see, the lines can get a little long.

We begin with the givens (lines 1 and 2), and express them logically (lines 3 and 4). We did not include "$d, e \in \mathcal{U}$" on lines 3 through 7 in the interest of space, instead opting to pre-declare them on the first line of the proof block.

The proof gets more interesting in line 5, where we combine the universal quantifications from lines 3 and 4. We can do this because a universal quantification is just that: Universal. Both $d \in A \to d \in B$ and $d \in B \to d \in C$ are assumed true for all elements, which we can say in one statement rather than two. We aren't required to combine them in order to complete the proof, but doing so allows us to show that such a merger is possible.

Line 6 differs from Line 5 by just one character: The last B is replaced by a C. Here's how we can justify making that replacement. We know, from Line 5, that there's an element in B ($\exists e(e \in B)$). We also know that if $e \in B$, then $e \in C$, by the universal quantification portion of Line 5. By Modus Ponens, we know that $e \in C$ follows.

At this point, the odds are good that you have at least one of these two questions: "Why do we have to make that one-letter change?", and "$e \in B$ is still assumed true, so why isn't it still in the expression of Line 6?" One answer covers both questions. We know our destination: $A \subset C$. To get there, we need to be able to assume that $\exists e(e \notin A \wedge e \in C)$. $e \in B$ is indeed still assumed to be true, but it's no longer useful to us, so we dropped it. Keeping it just adds unnecessary clutter to our already well-cluttered proof.

Line 7 neatly follows from Line 6, thanks to Hypothetical Syllogism on $d \in A \to d \in B$ and $d \in B \to d \in C$. The resulting expression is exactly the logical form of $A \subset C$. Making sure that the reader recognizes that completes the argument.

A common type of set proof requires showing that two set expressions S and T are equal; that is, that they describe the same set of elements. Here are two possible approaches to this type of conjecture:

1. Follow the definition of set equality: Prove that both $S \subseteq T$ and $T \subseteq S$ are true, using the logical expression for $\subseteq$.

2. Express one side in terms of set builder notation and logical operators, prove, and convert back to set notation.

The latter is usually less work, but the former is good to know, because it also shows how to prove subset (and proper subset) expressions.

To make the two approaches easier to compare and contrast, we will apply them both to the same set expression. This expression looks like it should be straight-forward to prove, and it is ... if you choose the right approach. We will start with the subset definition approach.

Example 119:

Problem: Prove or disprove: $S - S = \emptyset$.

Solution: Applying the definition of subset naturally leads to a two-part proof, one for each 'direction.'

Proof (Direct): By the definition of set equality, if $S - S \subseteq \emptyset$ and $\emptyset \subseteq S - S$, then $S - S = \emptyset$. We will prove both parts. Throughout, assume that $y, z \in \mathcal{U}$.

Case 1: Consider $S - S \subseteq \emptyset$.

$$
\begin{array}{lll}
(1) & S - S \subseteq \emptyset \;=\; \forall z(z \in S - S \rightarrow z \in \emptyset) & [\text{ Def. of } \subseteq] \\
(2) & =\; \forall z(z \in S - S \rightarrow \mathbf{F}) & [\text{ Nothing } \in \emptyset] \\
(3) & =\; \forall z(z \notin S - S) & [\text{ Law of False Cons. }] \\
(4) & =\; \forall z(z \notin \{y \mid y \in S \wedge y \notin S\}) & [\text{ Def. of Set Diff. }] \\
(5) & =\; \forall z(z \notin \{y \mid \mathbf{F}\}) & [\text{ Negation Laws }] \\
(6) & =\; \forall z(z \notin \emptyset) & [\text{ Meaning of } \emptyset] \\
(7) & =\; \mathbf{T} & [\text{ Still nothing } \in \emptyset!]
\end{array}
$$

Case 2: Consider $\emptyset \subseteq S - S$.

$$
\begin{array}{lll}
(1) & \emptyset \subseteq S - S \;=\; \forall z(z \in \emptyset \rightarrow z \in S - S) & [\text{ Def. of } \subseteq] \\
(2) & =\; \forall z(\mathbf{F} \rightarrow z \in S - S) & [\text{ Nothing } \in \emptyset] \\
(3) & =\; \forall z(\mathbf{T}) & [\text{ Def. of } \rightarrow] \\
(4) & =\; \mathbf{T} & [\text{ Tautology }]
\end{array}
$$

Therefore, $S - S = \emptyset$.

Several of the steps of this proof rely on knowledge of the empty set. For example, line 2 of Case 1 follows from the realization that the empty set contains no elements, meaning that $z \in \emptyset$ must be false. A less clear example from Case 1 is the step from line 5 to line 6. $\{y \mid \mathbf{F}\}$ looks cryptic, but can be thought of as a trivial case for set builder notation. Elements are in the set 'such that false.' That is, the condition for membership can never be true, and as no element can pass the inclusion test, the set must be empty.

Case 2 is much less complex. The trickiest part is interpreting $\forall z(\mathbf{T})$, but doing so is easier if you think about truth tables and tautologies. In a truth table, we can see that a logical expression is a tautology when the last column contains only values of 'true.' Similarly, $\forall z(\mathbf{T})$ says that the condition is true for all elements of the universe. That's as true as it gets. Note that we could have included the same step in Case 1 ($\ldots \forall z(z \notin \emptyset) = \forall z(\mathbf{T}) = \mathbf{T}$), but doing so wouldn't have clarified the argument.

If you didn't get too consumed by the details of the first case of the proof of Example 119, you may have noticed something interesting about the first case: Hidden inside is a proof of the original conjecture! Example 119's proof is fine, but it's far longer than the 'convert to logic' alternative, which is the version hidden in the first case. Here's that approach, as a stand-alone proof.

Example 120:

Problem: Prove or disprove: $S - S = \emptyset$.

Solution: As we've already created this approach as part of Example 119, we'll get straight to the proof.

Proof (Direct): Assume that $y \in \mathcal{U}$.

$$
\begin{array}{llll}
(1) & S - S & = \{y \mid y \in S \land y \notin S\} & [\text{ Def. of Set Difference }] \\
(2) & & = \{y \mid \mathbf{F}\} & [\text{ Negation Laws }] \\
(3) & & = \emptyset & [\text{ Meaning of } \emptyset]
\end{array}
$$

Therefore, $S - S = \emptyset$.

Short and sweet! This is clearly the technique to use to prove this conjecture, but Example 119 was much more educational.

These set proof examples have introduced a few new equalities and equivalences that are handy to remember when writing set proofs. Table 19 has a more complete list.

Table 19: Foundational Set Equalities and Equivalences

(a)	$\forall z(\mathbf{T}) \equiv \mathbf{T}$	Tautology
(b)	$\forall z(\mathbf{F}) \equiv \mathbf{F}$	Contradiction
(c)	$\{z \mid \mathbf{T}\} = \mathcal{U}$	"Such that true" means all elements are in the set
(d)	$\{z \mid \mathbf{F}\} = \emptyset$	"Such that false" means no elements are in the set

A reminder: Proof-writing is often a slow process that involves a lot of 'wasted' work, work that never appears in the final version that you show the world. That can be frustrating, but remember that this happens to everyone who writes proofs, paints, repairs cars, breathes, etc. The more proofs you write, the better you will get at selecting an initial proof approach, and the faster your brain will dig up useful answers to all of the "OK, *now* what do I do?" questions.

6.5   Choosing a Set Representation

A set can be viewed as an unordered list without duplicate elements. This naturally leads computer programmers to think of arrays, linked lists, trees, and hash tables as possible data structures for set representations.

An alternative way to look at the problem is to consider the operations that need to be performed on sets. This is the subject of one of our favorite programming mantras: "Choose the representation that best supports the operations." We are now very familiar with the typical set operations (and operation-like characteristics such as subset). The question: Do any of those classic linear data structures do a good job supporting set operations?

As this isn't a data structures book, you won't be subjected to an exhaustive analysis. Instead, we will give a quick answer: Not really. OK, we can be a little less quick. All of the data structures mentioned at the top of this section can be used, but they all have their own advantages and disadvantages, including operation efficiency (having an ordered representation makes some operations more efficient to execute) and space (references stored in linked list and tree nodes add storage overhead).

Example 121:

Java, starting with version 1.2, has offered a Set interface with none of the basic set operators . . . at least not by the names we know them. For example, `addAll()` and `removeAll()` are essentially union and difference, respectively, while `contains()` can be used to construct intersection.

This interface is supported by classes such as `HashSet` and `TreeSet`, which, as their names suggest, use a hash table and a tree (as of this writing a Red-Black tree, a kind of balanced binary search tree) as their representations. With this arrangement, Java allows programmers to select a representation that best fits their program's needs.

---

bit vector

A space-saving alternative representation for a set, one that also does a remarkably good job supporting set operators, is a *bit vector*. Essentially, a bit vector is just an array of bits. To use one as a set representation, we assign each member of the universe a position in the vector. If the bit is '0', the set doesn't contain the corresponding element; if it is '1', the set does contain it.

Example 122:

Consider $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. This means that we need a bit vector of nine bits in length. Let the left-most bit represent 1, the next 2,

---

etc. Using that mapping, here are some examples of sets and their corresponding bit vector representations.

Set	Representation
$\{3, 4\}$	001100000
$\{1, 3, 5, 7, 9\}$	101010101
$\emptyset$	000000000
$\mathcal{U}$	111111111

Are you concerned that, by using a bit vector, we are imposing an ordering on the elements of $\mathcal{U}$? We aren't changing the definition of a set; the ordering of the set elements is still irrelevant. $\{3, 4\}$ and $\{4, 3\}$ are still the same set. Both sets just have the same representation when viewed as a bit vector, which, as they are the same set, makes a lot of sense. Thus, the concept and the representation need not possess exactly the same characteristics. What matters is how well the representation supports the concept and its operations.

The second advantage of using bit vectors to represent sets is that most set operations can be performed very efficiently. Computer central processing units (CPUs) are designed to perform a relatively small collection of basic operations very efficiently. Those operations almost always include bit-wise logical operations such as AND and OR. A 64-bit CPU, for example, can AND or OR together two 64-bit values, which means we can perform set intersection and union operations on sets from a 64-element universe with a single CPU instruction. 64 elements may not seem like a lot, but that's more than the quantities of upper- and lower-case letters in the modern Latin alphabet (26 each), and the quantity of Roman digits (10), combined. Table 20 shows how the four basic set operations can be performed on bit vectors.

Table 20: Set and Corresponding Bit-wise Logical Operators

Set Operators	Bit-wise Logical Operators
$A \cup B$	$A \vee B$
$A \cap B$	$A \wedge B$
$\overline{A}$	$\overline{A}$
$A - B \ (= A \cap \overline{B})$	$A \wedge \overline{B}$

The fact that $A - B = A \cap \overline{B}$ was introduced in Appendix A. If you are curious how to perform these bit-wise logical operators in C or Java programs, see Table 6 in Chapter 1.

Example 123:

Again let $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Also assume that $O = \{1, 3, 5, 7, 9\}$, $E = \{2, 4, 6, 8\}$, and $C = \{4, 6, 8, 9\}$. As bit vectors, based on the given element ordering of $\mathcal{U}$, $O = $ `101010101`, $E = $ `010101010`, and $C = $ `000101011`.

The following tables demonstrate each of the four set operators. Union, intersection, and complement are straight-forward. Difference, because it is a compound operator, requires an additional step.

Union:

$$
\begin{array}{rl}
 & O \\
\cup & E \\
\hline
 & \mathcal{U}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \{1, 3, 5, 7, 9\} \\
\cup & \{2, 4, 6, 8\} \\
\hline
 & \{1, 2, 3, 4, 5, 6, 7, 8, 9\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \texttt{101010101} \\
\vee & \texttt{010101010} \\
\hline
 & \texttt{111111111}
\end{array}
$$

Intersection:

$$
\begin{array}{rl}
 & C \\
\cap & E \\
\hline
 & \{4, 6, 8\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \{4, 6, 8, 9\} \\
\cap & \{2, 4, 6, 8\} \\
\hline
 & \{4, 6, 8\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \texttt{000101011} \\
\wedge & \texttt{010101010} \\
\hline
 & \texttt{000101010}
\end{array}
$$

Complement:

$$
\begin{array}{c}
\overline{O} \\
\hline
E
\end{array}
\Rightarrow
\begin{array}{c}
\overline{\{1, 3, 5, 7, 9\}} \\
\hline
\{2, 4, 6, 8\}
\end{array}
\Rightarrow
\begin{array}{c}
\overline{\texttt{101010101}} \\
\hline
\texttt{010101010}
\end{array}
$$

Difference:

$$
\begin{array}{rl}
 & E \\
- & C \\
\hline
 & \{2\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & E \\
\cap & \overline{C} \\
\hline
 & \{2\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \{2, 4, 6, 8\} \\
\cap & \overline{\{4, 6, 8, 9\}} \\
\hline
 & \{2\}
\end{array}
\Rightarrow
\begin{array}{rl}
 & \texttt{010101010} \\
\wedge & \texttt{111010100} \\
\hline
 & \texttt{010000000}
\end{array}
$$

Bit vectors seem to be a great representation; they have both space efficiency (just 1 bit of storage per element) and operation efficiency (operators correspond to CPU instructions). Unfortunately, if we look more closely, problems appear, including:

1. We need to keep track of the mapping between the bit positions and the elements of the sets,

2. Not all operators are bit-vector-friendly (e.g., Cartesian Product), and

3. Subsets of infinite sets cannot be represented.

Choosing a data representation means accepting tradeoffs. For example, if we use a linked list as a set representation, we can store the set elements directly (no mapping to bits needed) and we can store a subset of an infinite set, but storing the sets will require much more space (to maintain the list structure) and processing the operators will be considerably more expensive (the lists must be traversed). Good software developers consider such issues before settling on a representation ... and they (usually grudgingly!) switch to a new representation when one is needed.