5 Infinity

5.1 Surjective and Injective Relations

There are a few properties of relations that will be useful when we take up the topic of counting because they imply certain relations between the *sizes* of domains and codomains. We say a binary relation $R : A \to B$ is:

- *total* when every element of A is assigned to some element of B; more concisely, R is total iff $A = RB$.
- *surjective* when every element of B is mapped to *at least once*[1]; more concisely, R is surjective iff $AR = B$.
- *injective* if every element of B is mapped to *at most once*, and
- *bijective* if R is total, surjective, and injective *function*.

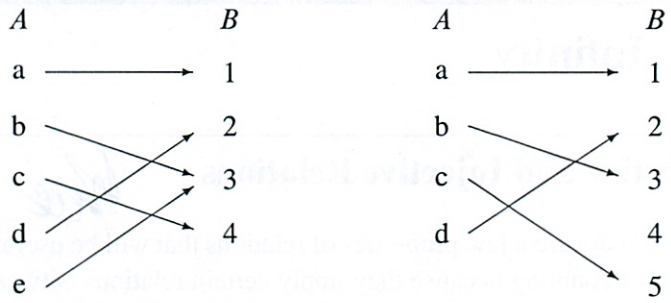
Note that this definition of R being total agrees with the definition in Section 4.3 when R is a function.

If R is a binary relation from A to B, we define AR to to be the *range* of R. So a relation is surjective iff its range equals its codomain. Again, in the case that R is a function, these definitions of "range" and "total" agree with the definitions in Section 4.3.

5.1.1 Relation Diagrams

We can explain all these properties of a relation $R : A \to B$ in terms of a diagram where all the elements of the domain, A, appear in one column (a very long one if A is infinite) and all the elements of the codomain, B, appear in another column, and we draw an arrow from a point a in the first column to a point b in the second column when a is related to b by R. For example, here are diagrams for two functions:

[1] The names "surjective" and "injective" are unmemorable and nondescriptive. Some authors use the term *onto* for surjective and *one-to-one* for injective, which are shorter but arguably no more memorable.



Here is what the definitions say about such pictures:

- "R is a function" means that every point in the domain column, A, has *at most one arrow out of it*.
- "R is total" means that *every* point in the A column has *at least one arrow out of it*. So if R is a function, being total really means every point in the A column has *exactly one arrow out of it*.
- "R is surjective" means that *every* point in the codomain column, B, has *at least one arrow into it*.
- "R is injective" means that every point in the codomain column, B, has *at most one arrow into it*.
- "R is bijective" means that *every* point in the A column has exactly one arrow out of it, and *every* point in the B column has exactly one arrow into it.

So in the diagrams above, the relation on the left is a total, surjective function (every element in the A column has exactly one arrow out, and every element in the B column has at least one arrow in), but not injective (element 3 has two arrows going into it). The relation on the right is a total, injective function (every element in the A column has exactly one arrow out, and every element in the B column has at most one arrow in), but not surjective (element 4 has no arrow going into it).

Notice that the arrows in a diagram for R precisely correspond to the pairs in the graph of R. But graph(R) does not determine by itself whether R is total or surjective; we also need to know what the domain is to determine if R is total, and we need to know the codomain to tell if it's surjective.

Example 5.1.1. The function defined by the formula $1/x^2$ is total if its domain is $\mathbb{R}^+$ but partial if its domain is some set of real numbers including 0. It is bijective if its domain and codomain are both $\mathbb{R}^+$, but neither injective nor surjective if its domain and codomain are both $\mathbb{R}$.

5.2 The Mapping Rule

The relational properties above are useful in figuring out the relative sizes of domains and codomains.

If A is a finite set, we let $|A|$ be the number of elements in A. A finite set may have no elements (the empty set), or one element, or two elements,... or any nonnegative integer number of elements.

Now suppose $R : A \to B$ is a function. Then every arrow in the diagram for R comes from exactly one element of A, so the number of arrows is at most the number of elements in A. That is, if R is a function, then

$$|A| \geq \#\text{arrows}.$$

Similarly, if R is surjective, then every element of B has an arrow into it, so there must be at least as many arrows in the diagram as the size of B. That is,

$$\#\text{arrows} \geq |B|.$$

Combining these inequalities implies that if R is a surjective function, then $|A| \geq |B|$. In short, if we write A surj B to mean that there is a surjective function from A to B, then we've just proved a lemma: if A surj B, then $|A| \geq |B|$. The following definition and lemma lists include this statement and three similar rules relating domain and codomain size to relational properties.

Definition 5.2.1. Let A, B be (not necessarily finite) sets. Then

1. A surj B iff there is a surjective *function* from A to B.
2. A inj B iff there is a total injective *relation* from A to B.
3. A bij B iff there is a bijection from A to B.
4. A strict B iff A surj B, but not B surj A.

Lemma 5.2.2. *[Mapping Rules] Let A and B be finite sets.*

1. *If A surj B, then $|A| \geq |B|$.*
2. *If A inj B, then $|A| \leq |B|$.*
3. *If R bij B, then $|A| = |B|$.*
4. *If R strict B, then $|A| > |B|$.*

Mapping rule 2 can be explained by the same kind of "arrow reasoning" we used for rule 1. Rules 3 and 4 are immediate consequences of these first two mapping rules.

5.3 The sizes of infinite sets

Mapping Rule 1 has a converse: if the size of a finite set, A, is greater than or equal to the size of another finite set, B, then it's always possible to define a surjective function from A to B. In fact, the surjection can be a total function. To see how this works, suppose for example that

$$A = \{a_0, a_1, a_2, a_3, a_4, a_5\}$$
$$B = \{b_0, b_1, b_2, b_3\}.$$

Then define a total function $f : A \to B$ by the rules

$$f(a_0) ::= b_0,\ f(a_1) ::= b_1,\ f(a_2) ::= b_2,\ f(a_3) = f(a_4) = f(a_5) ::= b_3.$$

In fact, if A and B are finite sets of the same size, then we could also define a bijection from A to B by this method.

In short, we have figured out if A and B are finite sets, then $|A| \geq |B|$ *if and only if* A surj B, and similar iff's hold for all the other Mapping Rules:

Lemma 5.3.1. *For* finite *sets, A, B,*

$$|A| \geq |B| \quad \textit{iff} \quad A \text{ surj } B,$$
$$|A| \leq |B| \quad \textit{iff} \quad A \text{ inj } B,$$
$$|A| = |B| \quad \textit{iff} \quad A \text{ bij } B,$$
$$|A| > |B| \quad \textit{iff} \quad A \text{ strict } B.$$

This lemma suggests a way to generalize size comparisons to infinite sets, namely, we can think of the relation surj as an "*at least as big as*" relation between sets, even if they are infinite. Similarly, the relation bij can be regarded as a "*same size*" relation between (possibly infinite) sets, and strict can be thought of as a "*strictly bigger* than" relation between sets.

Warning: We haven't, and won't, define what the "size" of an infinite is. The definition of infinite "sizes" is cumbersome and technical, and we can get by just fine without it. All we need are the "as big as" and "same size" relations, surj and bij, between sets.

But there's something else to watch out for. We've referred to surj as an "as big as" relation and bij as a "same size" relation on sets. Of course most of the "as big as" and "same size" properties of surj and bij on finite sets do carry over to infinite sets, but *some important ones don't* —as we're about to show. So you have to be

careful: don't assume that surj has any particular "as big as" property on *infinite* sets until it's been proved.

Let's begin with some familiar properties of the "as big as" and "same size" relations on finite sets that do carry over exactly to infinite sets:

Lemma 5.3.2. *For any sets, A, B, C,*

1. *A* surj *B and B* surj *C, implies A* surj *C.*
2. *A* bij *B and B* bij *C, implies A* bij *C.*
3. *A* bij *B implies B* bij *A.*

Lemma 5.3.2.1 and 5.3.2.2 follow immediately from the fact that compositions of surjections are surjections, and likewise for bijections, and Lemma 5.3.2.3 follows from the fact that the inverse of a bijection is a bijection. We'll leave a proof of these facts to Problem 5.1.

Another familiar property of finite sets carries over to infinite sets, but this time it's not so obvious:

Theorem 5.3.3 (Schröder-Bernstein)**.** *For any sets A, B, if A* surj *B and B* surj *A, then A* bij *B.*

That is, the Schröder-Bernstein Theorem says that if A is at least as big as B and conversely, B is at least as big as A, then A is the same size as B. Phrased this way, you might be tempted to take this theorem for granted, but that would be a mistake. For infinite sets A and B, the Schröder-Bernstein Theorem is actually pretty technical. Just because there is a surjective function $f : A \to B$ —which need not be a bijection —and a surjective function $g : B \to A$ —which also need not be a bijection —it's not at all clear that there must be a bijection $e : A \to B$. The idea is to construct e from parts of both f and g. We'll leave the actual construction to Problem 5.6.

Infinity is different

A basic property of finite sets that does *not* carry over to infinite sets is that adding something new makes a set bigger. That is, if A is a finite set and $b \notin A$, then $|A \cup \{b\}| = |A| + 1$, and so A and $A \cup \{b\}$ are not the same size. But if A is infinite, then these two sets *are* the same size!

Lemma 5.3.4. *Let A be a set and $b \notin A$. Then A is infinite iff A* bij *$A \cup \{b\}$.*

Proof. Since A is *not* the same size as $A \cup \{b\}$ when A is finite, we only have to show that $A \cup \{b\}$ *is* the same size as A when A is infinite.

That is, we have to find a bijection between $A \cup \{b\}$ and A when A is infinite. Here's how: since A is infinite, it certainly has at least one element; call it a_0. But since A is infinite, it has at least two elements, and one of them must not be equal to a_0; call this new element a_1. But since A is infinite, it has at least three elements, one of which must not equal a_0 or a_1; call this new element a_2. Continuing in the way, we conclude that there is an infinite sequence $a_0, a_1, a_2, \ldots, a_n, \ldots$ of different elements of A. Now it's easy to define a bijection $e : A \cup \{b\} \to A$:

$$\begin{aligned} e(b) &::= a_0, \\ e(a_n) &::= a_{n+1} && \text{for } n \in \mathbb{N}, \\ e(a) &::= a && \text{for } a \in A - \{b, a_0, a_1, \ldots\}. \end{aligned}$$

already has it?

■

A set, C, is *countable* iff its elements can be listed in order, that is, the distinct elements is A are precisely

$$c_0, c_1, \ldots, c_n, \ldots.$$

This means that if we defined a function, f, on the nonnegative integers by the rule that $f(i) ::= c_i$, then f would be a bijection from $\mathbb{N}$ to C. More formally,

Definition 5.3.5. A set, C, is *countably infinite* iff $\mathbb{N}$ bij C. A set is *countable* iff it is finite or countably infinite.

nice description

A small modification[2] of the proof of Lemma 5.3.4 shows that countably infinite sets are the "smallest" infinite sets, namely, if A is a countably infinite set, then A surj $\mathbb{N}$.

Since adding one new element to an infinite set doesn't change its size, it's obvious that neither will adding any *finite* number of elements. It's a common mistake to think that this proves that you can throw in countably infinitely many new elements. But just because it's ok to do something any finite number of times doesn't make it OK to do an infinite number of times. For example, starting from 3, you can add 1 any finite number of times and the result will be some integer greater than or equal to 3. But if you add add 1 a countably infinite number of times, you don't get an integer at all.

It turns out you really can add a countably infinite number of new elements to a countable set and still wind up with just a countably infinite set, but another argument is needed to prove this:

[2] See Problem 5.2

Lemma 5.3.6. *If A and B are countable sets, then so is $A \cup B$.*

Proof. Suppose the list of distinct elements of A is $a_0, a_1, \ldots$ and the list of B is $b_0, b_1, \ldots$. Then a list of all the elements in $A \cup B$ is just

$$a_0, b_0, a_1, b_1, \ldots a_n, b_n, \ldots. \tag{5.1}$$

Of course this list will contain duplicates if A and B have elements in common, but then deleting all but the first occurrences of each element in list (5.1) leaves a list of all the distinct elements of A and B. ■

5.3.1 Infinities in Computer Science

We've run into a lot of computer science students who wonder why they should care about infinite sets: any data set in a computer memory is limited by the size of memory, and since the universe appears to have finite size, there is a limit on the possible size of computer memory.

The problem with this argument is that universe-size bounds on data items are so big and uncertain (the universe seems to be getting bigger all the time), that it's simply not helpful to make use of possible bounds. For example, by this argument the physical sciences shouldn't assume that measurements might yield arbitrary real numbers, because there can only be a finite number of finite measurements in a universe of finite lifetime. What do you think scientific theories would look like without using the infinite set of real numbers?

Similary, in computer science, it simply isn't plausible that writing a program to add nonnegative integers with up to as many digits as, say, the stars in the sky (billions of galaxies each with billions of stars), would be any different than writing a program that would add any two integers no matter how many digits they had.

That's why basic programming data types like integers or strings, for example, can be defined without imposing any bound on the sizes of data items. Each datum of type `string` has only a finite number of letters, but there are an infinite number of data items of type `string`. When we then consider string procedures of type `string-->string`, not only are there an infinite number of such procedures, but each procedure generally behaves differently on different inputs, so that a single `string-->string` procedure may embody an infinite number of behaviors.

In short, an educated computer scientist can't get around having to understand infinite sets.

? How can you conclude this ?

Problems for Section 5.3

Class Problems

Problem 5.1.
Define a *surjection relation*, surj, on sets by the rule

Definition. A surj B iff there is a surjective **function** from A to B.

Define the *injection relation*, inj, on sets by the rule

Definition. A inj B iff there is a total injective *relation* from A to B.

(a) Prove that if A surj B and B surj C, then A surj C.

(b) Explain why A surj B iff B inj A.

(c) Conclude from (a) and (b) that if A inj B and B inj C, then A inj C.

Problem 5.2. (a) Several students felt the proof of Lemma 5.3.4 was worrisome, if not circular. What do you think?

(b) Use the proof of Lemma 5.3.4 to show that if A is an infinite set, then there is surjective function from A to $\mathbb{N}$, that is, every infinite set is "as big as" the set of nonnegative integers.

Problem 5.3.
Let $R : A \to B$ be a binary relation. Use an arrow counting argument to prove the following generalization of the Mapping Rule:

Lemma. *If R is a function, and $X \subseteq A$, then*

$$|X| \geq |XR|.$$

Problem 5.4.
Let $A = \{a_0, a_1, \ldots, a_{n-1}\}$ be a set of size n, and $B = \{b_0, b_1, \ldots, b_{m-1}\}$ a set of size m. Prove that $|A \times B| = mn$ by defining a simple bijection from $A \times B$ to the nonnegative integers from 0 to $mn - 1$.

Problem 5.5.
The rational numbers fill in all the spaces between the integers, so a first thought is that there must be more of them than the integers, but it's not true. In this problem you'll show that there are the same number of nonnegative rational as nonnegative integers. In short, the nonnegative rationals are countable.

(a) Describe a bijection between all the integers, $\mathbb{Z}$, and the nonnegative integers, $\mathbb{N}$.

(b) Define a bijection between the nonnegative integers and the set, $\mathbb{N} \times \mathbb{N}$, of all the ordered pairs of nonnegative integers:

$$\begin{array}{l}(0,0),(0,1),(0,2),(0,3),(0,4),\ldots\\(1,0),(1,1),(1,2),(1,3),(1,4),\ldots\\(2,0),(2,1),(2,2),(2,3),(2,4),\ldots\\(3.0),(3,1),(3,2),(3,3),(3,4),\ldots\\\vdots\end{array}$$

(c) Conclude that $\mathbb{N}$ is the same size as the set, $\mathbb{Q}$, of all nonnegative rational numbers.

Problem 5.6.
Suppose sets A and B have no elements in common, and

- A is as small as B because there is a total injective function $f : A \to B$, and
- B is as small as A because there is a total injective function $g : B \to A$.

Picturing the diagrams for f and g, there is *exactly one* arrow *out* of each element—a left-to-right f-arrow if the element in A and a right-to-left g-arrow if the element in B. This is because f and g are total functions. Also, there is *at most one* arrow *into* any element, because f and g are injections.

So starting at any element, there is a unique, and unending path of arrows going forwards. There is also a unique path of arrows going backwards, which might be unending, or might end at an element that has no arrow into it. These paths are completely separate: if two ran into each other, there would be two arrows into the element where they ran together.

This divides all the elements into separate paths of four kinds:

i. paths that are infinite in both directions,

ii. paths that are infinite going forwards starting from some element of A.

iii. paths that are infinite going forwards starting from some element of B.

iv. paths that are unending but finite.

(a) What do the paths of the last type (iv) look like?

(b) Show that for each type of path, either

- the f-arrows define a bijection between the A and B elements on the path, or
- the g-arrows define a bijection between B and A elements on the path, or
- both sets of arrows define bijections.

For which kinds of paths do both sets of arrows define bijections?

(c) Explain how to piece these bijections together to prove that A and B are the same size.

Homework Problems

Problem 5.7.

Let $f : A \to B$ and $g : B \to C$ be functions and $h : A \to C$ be their composition, namely, $h(a) ::= g(f(a))$ for all $a \in A$.

(a) Prove that if f and g are surjections, then so is h.

(b) Prove that if f and g are bijections, then so is h.

(c) If f is a bijection, then define $f' : B \to A$ so that

$$f'(b) ::= \text{ the unique } a \in A \text{ such that } f(a) = b.$$

Prove that f' is a bijection. (The function f' is called the *inverse* of f. The notation f^{-1} is often used for the inverse of f.)

Problem 5.8.

In this problem you will prove a fact that may surprise you—or make you even more convinced that set theory is nonsense: the half-open unit interval is actually the *same size* as the nonnegative quadrant of the real plane![3] Namely, there is a bijection from $(0, 1]$ to $[0, \infty)^2$.

(a) Describe a bijection from $(0, 1]$ to $[0, \infty)$.

Hint: $1/x$ almost works.

[3]The half open unit interval, $(0, 1]$, is $\{r \in \mathbb{R} \mid 0 < r \leq 1\}$. Similarly, $[0, \infty) ::= \{r \in \mathbb{R} \mid r \geq 0\}$.

(b) An infinite sequence of the decimal digits $\{0, 1, \ldots, 9\}$ will be called *long* if it has infinitely many occurrences of some digit other than 0. Let L be the set of all such long sequences. Describe a bijection from L to the half-open real interval $(0, 1]$.

Hint: Put a decimal point at the beginning of the sequence.

(c) Describe a surjective function from L to L^2 that involves alternating digits from two long sequences. a *Hint:* The surjection need not be total.

(d) Prove the following lemma and use it to conclude that there is a bijection from L^2 to $(0, 1]^2$.

Lemma 5.3.7. *Let A and B be nonempty sets. If there is a bijection from A to B, then there is also a bijection from $A \times A$ to $B \times B$.*

(e) Conclude from the previous parts that there is a surjection from $(0, 1]$ and $(0, 1]^2$. Then appeal to the Schröder-Bernstein Theorem to show that there is actually a bijection from $(0, 1]$ and $(0, 1]^2$.

(f) Complete the proof that there is a bijection from $(0, 1]$ to $[0, \infty)^2$.

Larger Infinities

There are lots of different sizes of infinite sets. For example, starting with the infinite set, $\mathbb{N}$, of nonnegative integers, we can build the infinite sequence of sets

$$\mathbb{N} \text{ strict } \mathcal{P}(\mathbb{N}) \text{ strict } \mathcal{P}(\mathcal{P}(\mathbb{N})) \text{ strict } \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))) \text{ strict } \ldots.$$

By Theorem 5.2.6, each of these sets is strictly bigger than all the preceding ones. But that's not all: the union of all the sets in the sequence is strictly bigger than each set in the sequence (see Problem 5.7). In this way you can keep going indefinitely, building "bigger" infinities all the way.

Additional section

5.3 The Halting Problem

Granted that towers of larger and larger infinite sets is at best just a romantic concern for a computer scientist, the *reasoning* that leads to these conclusions plays a critical role in the theory of computation. Cantor's proof embodies the simplest form of what is known as a "diagonal argument." Diagonal arguments are used to establish show that lots of problems logically just can't be solved by computation, and there is no getting around it.

This story begins with a reminder that having procedures operate on programs is a basic part of computer science technology. For example, *compilation* refers to taking any given program text written in some "high level" programming language like Java, C++, Python, ..., and then generating a program of low-level instructions that does the same thing but is targeted to run well on available hardware. Similarly, *interpreters* or *virtual machines* are procedures that take a program text designed to be run on one kind of computer and simulate it on another kind of computer. Routine features of compilers involve "type-checking" programs to ensure that certain kinds of run-time errors won't happen, and "optimizing" the generated programs so they run faster or use less memory.

Now the fundamental thing that computation logically just can't do is a *perfect* job of type-checking, optimizing, or any kind of analysis of the complete run time behavior of programs. In this section we'll illustrate this with a basic example known as the *Halting Problem*. The general Halting Problem for some programming language is, given an arbitrary program, recognize when running the program will not finish successfully —halt —because it aborts with some kind of error, or because it simply never stops. Of course it's easy to detect when any given program *will* halt: just run it on a virtual machine and wait. The problem is what if the given program does *not* halt —how do you recognize that? We will use a diagonal argu-

ment to prove that if an analysis program tries to recognize non-halting programs, it is bound to give wrong answers, or no answers, for an infinite number of programs it might have to analyze!

To be precise about this, let's call a programming procedure —written in your favorite programming language such C++, or Java, or Python —a *string procedure* when it is applicable to strings over a standard alphabet —say the 256 character ASCII alphabet. When a string procedure applied to an ASCII string returns the boolean value **True**, we'll say the procedure *recognizes* the string. If the prccedure does anything else —returns a value other than **True**, aborts with an error, runs forever,...—then it doesn't recognize the string.

As a simple example, you might think about how to write a string procedure that recognizes precisely those *double letter* strings where every character occurs twice in a row. For example, `aaCC33`, and `zz++ccBB` are double letter ASCII strings, but textttaa;bb, `b33`, and `AAAAA` are not. Even better, how about actually writing a recognizer for the double letter ASCII strings in your favorite programming language?

We'll call a set of strings *recognizable* if there is a procedure that recognizes precisely that set of strings. So the set of double letter strings is recognizable.

There is no harm in assuming that every program can be written using only the ASCII characters; they usually are anyway. When an ASCII string, s, is actually the ASCII description of some string procedure, we'll refer to that string procedure as P_s. You can think of P_s as the result of compiling s.[3] It's technically helpful to treat *every* ASCII string as a program for a string procedure. So when a string, s, doesn't parse as a proper string procedure, we'll define P_s to be some default string procedure —say one that always returns **False**.

Now can define the precise set of strings that describe non-halting programs:

Definition 5.3.1.

$$\text{No-halt} ::= \{s \mid s \text{ is an ASCII string and } P_s \text{ does not recognize } s\}. \tag{5.6}$$

Recognizing the strings in No-halt is a special case of the Halting Problem. We'll blow way any chance of having a program solve the general problem by showing that no program can solve this special case. In particular, we're going to prove

Theorem 5.3.2. *No-halt is not recognizable.*

We'll use an argument just like Cantor's in the proof of Theorem 5.2.6.

[3]The string, s, and the procedure, P_s, have to be distinguished to avoid a type error: you can't apply a string to string. For example, let s be the string that you wrote as your program to recognize the double letter strings. Applying s to a string argument, say `aabbccdd`, should throw a type exception; what you need to do is compile s to the procedure P_s and then apply P_s to `aabbccdd`.

Proof. Namely, let $\mathcal{S}$ be the set of ASCII strings, and for any string $s \in \mathcal{S}$, let $f(s)$ be the set of strings recognized by P_s:

$$f(s) ::= \{t \in \mathcal{S} \mid P_s \text{ recognizes } t\}.$$

By convention, we associated a string procedure, P_s, with every string, $s \in \mathcal{S}$, which makes f a total function, and by definition,

$$s \in \text{No-halt} \quad \text{iff} \quad s \notin f(s), \tag{5.7}$$

for all strings, $s \in \mathcal{S}$.

Now suppose to the contrary that No-halt was recognizable. This means there is some procedure P_{s_0} that recognizes No-halt, which is the same as saying that

$$\text{No-halt} = f(s_0).$$

Combined with (5.7), we get

$$s \in f(s_0) \quad \text{iff} \quad s \notin f(s) \tag{5.8}$$

for all $s \in \mathcal{S}$. Now letting $s = s_0$ in (5.8) yields the immediate contradiction

$$s_0 \in f(s_0) \quad \text{iff} \quad s_0 \notin f(s_0).$$

This contradiction implies that No-halt cannot be recognized by any string procedure. ■

So that does it: it's logically impossible for programs in any particular language to solve just this special case of the general Halting Problem for programs in that language. And having proved that it's impossible to have a procedure that figures out whether a arbitrary program returns **True**, it's easy to show that it's impossible to have a procedure that is a perfect recognizer for *any* complete run time property of programs.

For example, most compilers do "static" type-checking at compile time to ensure that programs won't make run-time type errors. A program that type-checks is guaranteed not to cause a run-time type-error. But since it's impossible to recognize perfectly when programs won't cause type-errors, it follows that the type-checker must be rejecting programs that really wouldn't cause a type-error. The conclusion is that no type-checker is perfect —you can always do better!

It's a different story if we think about the *practical* possibility of writing programming analyzers. The fact that it's logically impossible to analyze perfectly arbitrary programs does not mean that you can't do a very good job analyzing interesting programs that come up in practice. In fact these "interesting" programs are

commonly *intended* to be analyzable in order to confirm that they do what they're supposed to do.

So it's not clear how much of a hurdle this theoretical limitation implies in practice. What the theory does provide is some perspective on claims about general analysis methods for programs. The theory tells us that people who make such claims either

- are exaggerating the power (if any) of their methods—say to make a sale or get a grant. or
- are trying to keep things simple by not going into technical limitations they're aware of, or
- perhaps most commonly, are so excited about some useful practical successes of their methods that they haven't bothered to think about the limitations which you know must be there.

So from now on, if you hear people making claims about having general program analysis/verification/optimization methods, you'll know they can't be telling the whole story.

One more important point: there's no hope of getting around this by switching programming languages. Our proof covered programs written in some given programming language like Java, for example, and concluded that no Java program can perfectly analyze all Java programs. Could there be a C++ analysis procedure that successfully takes on all Java programs? After all, C++ does allow more intimate manipulation of computer memory than Java does. But there is no loophole here: it's possible to write a virtual machine for C++ in Java, so if there were a C++ procedure that analyzed Java programs, the Java virtual machine would be able to do it too, and that's impossible. These logical limitations on the power of computation apply no matter what kinds of programs or computers you use.

5.4 The Logic of Sets

5.4.1 Russell's Paradox

Reasoning naively about sets turns out to be risky. In fact, one of the earliest attempts to come up with precise axioms for sets in the late nineteenth century by the logician Gotlob Frege, was shot down by a three line argument known as *Rus-*

6 First-Order Logic

6.1 The Logic of Sets

6.1.1 Russell's Paradox

Reasoning naively about sets turns out to be risky. In fact, one of the earliest attempts to come up with precise axioms for sets by a late nineteenth century logican named Gotlob *Frege* was shot down by a three line argument known as *Russell's Paradox*:[1] This was an astonishing blow to efforts to provide an axiomatic foundation for mathematics.

> Let S be a variable ranging over all sets, and define
>
> $$W ::= \{S \mid S \notin S\}.$$
>
> So by definition,
>
> $$S \in W \text{ iff } S \notin S,$$
>
> for every set S. In particular, we can let S be W, and obtain the contradictory result that
>
> $$W \in W \text{ iff } W \notin W.$$

That's what I would think

A way out of the paradox was clear to Russell and others at the time: *it's unjustified to assume that W is a set.* So the step in the proof where we let S be W has no justification, because S ranges over sets, and W may not be a set. In fact, the paradox implies that W had better not be a set!

But denying that W is a set means we must *reject* the very natural axiom that every mathematically well-defined collection of elements is actually a set. So the problem faced by Frege, Russell and their colleagues was how to specify *which*

[1] Bertrand *Russell* was a mathematician/logician at Cambridge University at the turn of the Twentieth Century. He reported that when he felt too old to do mathematics, he began to study and write about philosophy, and when he was no longer smart enough to do philosophy, he began writing about politics. He was jailed as a conscientious objector during World War I. For his extensive philosophical and political writing, he won a Nobel Prize for Literature.

well-defined collections are sets. Russell and his fellow Cambridge University colleague Whitehead immediately went to work on this problem. They spent a dozen years developing a huge new axiom system in an even huger monograph called *Principia Mathematica*.

6.1.2 The ZFC Axioms for Sets

It's generally agreed that, using some simple logical deduction rules, essentially all of mathematics can be derived from some axioms about sets called the Axioms of Zermelo-Frankel Set Theory with Choice (ZFC).

We're *not* going to be working with these axioms in this course, but we thought you might like to see them –and while you're at it, get some practice reading quantified formulas:

Extensionality. Two sets are equal if they have the same members. In formal logical notation, this would be stated as:

$$(\forall z.\ (z \in x \text{ IFF } z \in y)) \text{ IMPLIES } x = y.$$

Pairing. For any two sets x and y, there is a set, $\{x, y\}$, with x and y as its only elements:

$$\forall x, y.\ \exists u.\ \forall z.\ [z \in u \text{ IFF } (z = x \text{ OR } z = y)]$$

Union. The union, u, of a collection, z, of sets is also a set:

$$\forall z.\ \exists u \forall x.\ (\exists y.\ x \in y \text{ AND } y \in z) \text{ IFF } x \in u.$$

Infinity. There is an infinite set. Specifically, there is a nonempty set, x, such that for any set $y \in x$, the set $\{y\}$ is also a member of x.

Power Set. All the subsets of a set form another set:

$$\forall x.\ \exists p.\ \forall u.\ u \subseteq x \text{ IFF } u \in p.$$

Replacement. Suppose a formula, ϕ, of set theory defines the graph of a function, that is,

$$\forall x, y, z.\ [\phi(x, y) \text{ AND } \phi(x, z)] \text{ IMPLIES } y = z.$$

Then the image of any set, s, under that function is also a set, t. Namely,

$$\forall s\ \exists t\ \forall y.\ [\exists x.\ \phi(x, y) \text{ IFF } y \in t].$$

Foundation. There cannot be an infinite sequence

$$\cdots \in x_n \in \cdots \in x_1 \in x_0$$

of sets each of which is a member of the previous one. This is equivalent to saying every nonempty set has a "member-minimal" element. Namely, define

$$\text{member-minimal}(m, x) ::= [m \in x \text{ AND } \forall y \in x.\, y \notin m].$$

Then the Foundation axiom is

$$\forall x.\, x \neq \emptyset \text{ IMPLIES } \exists m.\, \text{member-minimal}(m, x).$$

Choice. Given a set, s, whose members are nonempty sets no two of which have any element in common, then there is a set, c, consisting of exactly one element from each set in s.

6.1.3 Avoiding Russell's Paradox

These modern ZFC axioms for set theory are much simpler than the system Russell and Whitehead first came up with to avoid paradox. In fact, the ZFC axioms are as simple and intuitive as Frege's original axioms, with one technical addition: the Foundation axiom. Foundation captures the intuitive idea that sets must be built up from "simpler" sets in certain standard ways. And in particular, Foundation implies that no set is ever a member of itself. So the modern resolution of Russell's paradox goes as follows: since $S \notin S$ for all sets S, it follows that W, defined above, contains every set. This means W can't be a set —or it would be a member of itself.

Use this example to think about logic and proofs

nice solution

6.1.4 Power sets are strictly bigger

It turns out that the ideas behind Russell's Paradox, which caused so much trouble for the early efforts to formulate Set Theory, lead to a correct and astonishing fact about infinite sets: they are *not all the same size*.

In particular,

Theorem 6.1.1. *For any set, A, the power set, $\mathcal{P}(A)$, is strictly bigger than A.*

Proof. First of all, $\mathcal{P}(A)$ is as big as A: for example, the partial function $f : \mathcal{P}(A) \to A$, where $f(\{a\}) ::= a$ for $a \in A$ and f is only defined on one-element sets, is a surjection.

To show that $\mathcal{P}(A)$ is strictly bigger than A, we have to show that if g is a function from A to $\mathcal{P}(A)$, then g is not a surjection. So, mimicking Russell's Paradox, define

$$A_g ::= \{a \in A \mid a \notin g(a)\}.$$

Now A_g is a well-defined subset of A, which means it is a member of $\mathcal{P}(A)$. But A_g can't be in the range of g, because if it were, we would have

$$A_g = g(a_0)$$

for some $a_0 \in A$, so by definition of A_g,

$$a \in g(a_0) \quad \text{iff} \quad a \in A_g \quad \text{iff} \quad a \notin g(a)$$

for all $a \in A$. Now letting $a = a_0$ yields the contradiction

$$a_0 \in g(a_0) \quad \text{iff} \quad a_0 \notin g(a_0).$$

So g is not a surjection, because there is an element in the power set of A, namely the set A_g, that is not in the range of g. ■

Larger Infinities

There are lots of different sizes of infinite sets. For example, starting with the infinite set, $\mathbb{N}$, of nonnegative integers, we can build the infinite sequence of sets

$$\mathbb{N},\ \mathcal{P}(\mathbb{N}),\ \mathcal{P}(\mathcal{P}(\mathbb{N})),\ \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))),\ \ldots.$$

but I thought you can't add to it

By Theorem 6.1.1, each of these sets is strictly bigger than all the preceding ones. But that's not all: the union of all the sets in the sequence is strictly bigger than each set in the sequence (see Problem 6.1). In this way you can keep going, building still bigger infinities.

So there is an endless variety of different size infinities.

6.1.5 Does All This Really Work?

So this is where mainstream mathematics stands today: there is a handful of ZFC axioms from which virtually everything else in mathematics can be logically derived. This sounds like a rosy situation, but there are several dark clouds, suggesting that the essence of truth in mathematics is not completely resolved.

- The ZFC axioms weren't etched in stone by God. Instead, they were mostly made up by some guy named Zermelo. Probably some days he forgot his house keys.

So maybe Zermelo, just like Frege, didn't get his axioms right and will be shot down by some successor to Russell who will use his axioms to prove a proposition P and its negation NOT P. Then math would be broken. This sounds crazy, but after all, it has happened before.

In fact, while there is broad agreement that the ZFC axioms are capable of proving all of standard mathematics, the axioms have some further consequences that sound paradoxical. For example, the Banach-Tarski Theorem says that, as a consequence of the Axiom of Choice, a solid ball can be divided into six pieces and then the pieces can be rigidly rearranged to give *two* solid balls, each the same size as the original!

- Georg *Cantor* was a contemporary of Frege and Russell who first developed the theory of infinite sizes (because he thought he needed it in his study of Fourier series). Cantor raised the question whether there is a set whose size is strictly between the "smallest[2]" infinite set, $\mathbb{N}$, and $\mathcal{P}(\mathbb{N})$; he guessed not:

  Cantor's *Continuum Hypothesis*: There is no set, A, such that $\mathcal{P}(\mathbb{N})$ is strictly bigger than A and A is strictly bigger than $\mathbb{N}$.

  The Continuum Hypothesis remains an open problem a century later. Its difficulty arises from one of the deepest results in modern Set Theory — discovered in part by Gödel in the 1930's and Paul Cohen in the 1960's —namely, the ZFC axioms are not sufficient to settle the Continuum Hypothesis: there are two collections of sets, each obeying the laws of ZFC, and in one collection the Continuum Hypothesis is true, and in the other it is false. So settling the Continuum Hypothesis requires a new understanding of what Sets should be to arrive at persuasive new axioms that extend ZFC and are strong enough to determine the truth of the Continuum Hypothesis one way or the other.

- But even if we use more or different axioms about sets, there are some unavoidable problems. In the 1930's, Gödel proved that, assuming that an axiom system like ZFC is consistent —meaning you can't prove both P and NOT P for any proposition, P —then the very proposition that the system is consistent (which is not too hard to express as a logical formula) cannot be proved in the system. In other words, no consistent system is strong enough to verify itself.

[2] See Problem 5.2

This is like one of those perspective Stairwell Paintings

This is interesting - but way too technical

6.1.6 Large Infinities in Computer Science

If the romance of different size infinities and continuum hypotheses doesn't appeal to you, not knowing about them is not going to lower your professional abilities as a computer scientist. These abstract issues about infinite sets rarely come up in mainstream mathematics, and they don't come up at all in computer science, where the focus is generally on "countable," and often just finite, sets. In practice, only logicians and set theorists have to worry about collections that are too big to be sets. In fact, at the end of the 19th century, the general mathematical community doubted the relevance of what they called "Cantor's paradise" of unfamiliar sets of arbitrary infinite size.

But the proof that power sets are bigger gives the simplest form of what is known as a "diagonal argument." Diagonal arguments are used to prove many fundamental results about the limitations of computation, such as the undecidability of the Halting Problem for programs (see Problem 6.2) and the inherent, unavoidable, inefficiency (exponential time or worse) of procedures for other computational problems. So computer scientists do need to study diagonal arguments in order to understand the logical limits of computation.

6.2 Glossary of Symbols

symbol	meaning
$::=$	is defined to be
$\wedge$	and
$\vee$	or
$\longrightarrow$	implies
$\neg$	not
$\neg P$	not P
$\overline{P}$	not P
$\longleftrightarrow$	iff
$\longleftrightarrow$	equivalent
$\oplus$	xor
$\exists$	exists
$\forall$	for all
$\in$	is a member of
$\subseteq$	is a subset of
$\subset$	is a proper subset of
$\cup$	set union
$\cap$	set intersection
$\overline{A}$	complement of a set, A
$\mathcal{P}(A)$	powerset of a set, A
$\emptyset$	the empty set, $\{\}$

Problems for Section 6.1

Class Problems

Problem 6.1.

There are lots of different sizes of infinite sets. For example, starting with the infinite set, $\mathbb{N}$, of nonnegative integers, we can build the infinite sequence of sets

$$\mathbb{N},\ \mathcal{P}(\mathbb{N}),\ \mathcal{P}(\mathcal{P}(\mathbb{N})),\ \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))),\ \ldots.$$

By Theorem 6.1.1 from the Notes, each of these sets is *strictly bigger*[3] than all the preceding ones. But that's not all: if we let U be the union of the sequence of sets above, then U is strictly bigger than every set in the sequence! Prove this:

[3]Reminder: set A is *strictly bigger* than set B just means that A surj B, but NOT(B surj A).

Lemma. *Let $\mathcal{P}^n(\mathbb{N})$ be the nth set in the sequence, and*

$$U ::= \bigcup_{n=0}^{\infty} \mathcal{P}^n(\mathbb{N}).$$

Then

1. *U surj $\mathcal{P}^n(\mathbb{N})$ for every $n \in \mathbb{N}$, but*
2. *there is no $n \in \mathbb{N}$ for which $\mathcal{P}^n(\mathbb{N})$ surj U.*

Now of course, we could take $U, \mathcal{P}(U), \mathcal{P}(\mathcal{P}(U)), \ldots$ and can keep on indefinitely building still bigger infinities.

Problem 6.2.

Let's refer to a programming procedure (written in your favorite programming language—C++, or Java, or Python, ...) as a *string procedure* when it is applicable to data of type `string` and only returns values of type `boolean`. When a string procedure, P, applied to a `string`, s, returns **True**, we'll say that P *recognizes* s. If $\mathcal{R}$ is the set of strings that P recognizes, we'll call P a *recognizer* for $\mathcal{R}$.

(a) Describe how a recognizer would work for the set of strings containing only lower case Roman letter—`a,b,...,z`—such that each letter occurs twice in a row. For example, `aaccaabbzz`, is such a string, but `abb`, `00bb`, `AAbb`, and `a` are not. (Even better, actually write a recognizer procedure in your favorite programming language).

A set of `strings` is called *recognizable* if there is a recognizer procedure for it.

When you actually program a procedure, you have to type the program text into a computer system. This means that every procedure is described by some `string` of typed characters. If a `string`, s, is actually the typed description of some string procedure, let's refer to that procedure as P_s. You can think of P_s as the result of compiling s.[4]

In fact, it will be helpful to associate every string, s, with a procedure, P_s; we can do this by defining P_s to be some fixed string procedure—it doesn't matter which one—whenever s is not the typed description of an actual procedure that can

[4]The string, s, and the procedure, P_s, have to be distinguished to avoid a type error: you can't apply a string to string. For example, let s be the string that you wrote as your program to answer part (a). Applying s to a string argument, say `oorrmm`, should throw a type exception; what you need to do is apply the procedure P_s to `oorrmm`. This should result in a returned value **True**, since `oorrmm` consists of three pairs of lowercase roman letters

be applied to string s. The result of this is that we have now defined a total function, f, mapping every string, s, to the set, $f(s)$, of strings recognized by P_s. That is we have a total function,

$$f : \texttt{string} \to \mathcal{P}(\texttt{string}). \tag{6.1}$$

(b) Explain why the actual range of f is the set of all recognizable sets of strings.

This is exactly the set up we need to apply the reasoning behind Russell's Paradox to define a set that is not in the range of f, that is, a set of strings, $\mathcal{N}$, that is *not* recognizable.

(c) Let

$$\mathcal{N} ::= \{s \in \texttt{string} \mid s \notin f(s)\}.$$

Prove that $\mathcal{N}$ is not recognizable.

Hint: Similar to Russell's paradox or the proof of Theorem 6.1.1.

(d) Discuss what the conclusion of part (c) implies about the possibility of writing "program analyzers" that take programs as inputs and analyze their behavior.

Problem 6.3.

Though it was a serious challenge for set theorists to overcome Russells' Paradox, the idea behind the paradox led to some important (and correct :-)) results in logic and computer science.

To show how the idea applies, let's recall the formulas from Problem 3.13 that made assertions about binary strings. For example, one of the formulas in that problem was

$$\text{NOT}[\exists y\, \exists z.s = y1z] \tag{all-0s}$$

This formula defines a property of a binary string, s, namely that s has no occurrence of a 1. In other words, s is a string of (zero or more) 0's. So we can say that this formula *describes* the set of strings of 0's.

More generally, when G is any formula that defines a string property, let ok-strings(G) be the set of all the strings that have this property. A set of binary strings that equals ok-strings(G) for some G is called a *describable* set of strings. So, for example, the set of all strings of 0's is describable because it equals ok-strings(**all-0s**).

Now let's shift gears for a moment and think about the fact that formula **all-0s** appears above. This happens because instructions for formatting the formula were generated by a computer text processor (for this text, we used the LaTeX text processing system), and then an image suitable for printing or display was constructed

according to these instructions. Since everybody knows that data is stored in computer memory as binary strings, this means there must have been some binary string in computer memory—call it $t_{\textbf{all-0s}}$—that enabled a computer to display formula **all-0s** once $t_{\textbf{all-0s}}$ was retrieved from memory.

In fact, it's not hard to find ways to represent *any* formula, G, by a corresponding binary word, t_G, that would allow a computer to reconstruct G from t_G. We needn't be concerned with how this reconstruction process works; all that matters for our purposes is that every formula, G, has a representation as binary string, t_G.

Now let

$$V ::= \{t_G \mid G \text{ defines a property of strings and } t_G \notin \text{ok-strings}(G)\}.$$

Use reasoning similar to Russell's paradox to show that V is not describable.

Homework Problems

Problem 6.4.

Let $[\mathbb{N} \to \{1,2,3\}]$ be the set of all sequences containing only the numbers 1, 2, and 3, for example,

$$(1,1,1,1...),$$
$$(2,2,2,2...),$$
$$(3,2,1,3...).$$

For any sequence, s, let $s[m]$ be its mth element.

Prove that $[\mathbb{N} \to \{1,2,3\}]$ is uncountable.

Hint: Suppose there was a list

$$\mathcal{L} = \text{sequence}_0, \text{sequence}_1, \text{sequence}_2, \ldots$$

of sequences in $[\mathbb{N} \to \{1,2,3\}]$ and show that there is a "diagonal" sequence $\text{diag} \in [\mathbb{N} \to \{1,2,3\}]$ that does not appear in the list. Namely,

$$\text{diag} ::= r(\text{sequence}_0[0]), r(\text{sequence}_1[1]), r(\text{sequence}_2[2]), \ldots,$$

where $r : \{1,2,3\} \to \{1,2,3\}$ is some function such that $r(i) \neq i$ for $i = 1,2,3$.

Problem 6.5.

For any sets, A, and B, let $[A \to B]$ be the set of total functions from A to B. Prove that if A is not empty and B has more than one element, then NOT(A surj $[A \to B]$).

Hint: Suppose there is a function, σ, that maps each element $a \in A$ to a function $\sigma_a : A \to B$. Pick any two elements of B; call them 0 and 1. Then define

$$\text{diag}(a) ::= \begin{cases} 0 \text{ if } \sigma_a(a) = 1, \\ 1 \text{ otherwise.} \end{cases}$$

later printout

6 Induction

↑ repeating chap #

Induction is a powerful method for showing a property is true for all natural numbers. Induction plays a central role in discrete mathematics and computer science, and in fact, its use is a defining characteristic of *discrete* —as opposed to *continuous* —mathematics. This chapter introduces two versions of induction —Ordinary and Strong —and explains why work and how to use them in proofs. It also introduces the Invariance Principle, which is a version of induction specially adapted for reasoning about step-by-step processes.

good explanation

6.1 Ordinary Induction

To understand how induction works, suppose there is a professor who brings to class a bottomless bag of assorted miniature candy bars. She offers to share the candy in the following way. First, she lines the students up in order. Next she states two rules:

1. The student at the beginning of the line gets a candy bar.
2. If a student gets a candy bar, then the following student in line also gets a candy bar.

Let's number the students by their order in line, starting the count with 0, as usual in computer science. Now we can understand the second rule as a short description of a whole sequence of statements:

- If student 0 gets a candy bar, then student 1 also gets one.
- If student 1 gets a candy bar, then student 2 also gets one.
- If student 2 gets a candy bar, then student 3 also gets one.

⋮

Of course this sequence has a more concise mathematical description:

> If student n gets a candy bar, then student $n + 1$ gets a candy bar, for all nonnegative integers n.

nicer

So suppose you are student 17. By these rules, are you entitled to a miniature candy bar? Well, student 0 gets a candy bar by the first rule. Therefore, by the second rule, student 1 also gets one, which means student 2 gets one, which means student 3 gets one as well, and so on. By 17 applications of the professor's second rule, you get your candy bar! Of course the rules actually guarantee a candy bar to *every* student, no matter how far back in line they may be.

6.1.1 A Rule for Ordinary Induction

The reasoning that led us to conclude that every student gets a candy bar is essentially all there is to induction.

The Principle of Induction.

Let P be a predicate on nonnegative integers. If

- $P(0)$ is true, and
- $P(n)$ IMPLIES $P(n+1)$ for all nonnegative integers, n,

then

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

Since we're going to consider several useful variants of induction in later sections, we'll refer to the induction method described above as *ordinary induction* when we need to distinguish it. Formulated as a proof rule, this would be

Rule. ***Induction Rule***

$$\frac{P(0), \quad \forall n \in \mathbb{N}.\ P(n) \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\ P(m)}$$

This general induction rule works for the same intuitive reason that all the students get candy bars, and we hope the explanation using candy bars makes it clear why the soundness of the ordinary induction can be taken for granted. In fact, the rule is so obvious that it's hard to see what more basic principle could be used to justify it.[1] What's not so obvious is how much mileage we get by using it.

6.1.2 A Familiar Example

The formula (6.1) below for the sum of the nonnegative integers up to n is the kind of statement about all nonnegative integers to which induction applies directly. We

[1] But see Section 6.4.

already already proved it (Theorem 2.3.1) using the Well Ordering Principle, but now we'll prove it using induction.

Theorem. *For all $n \in \mathbb{N}$,*

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \tag{6.1}$$

To use the Induction Principle to prove the Theorem, define predicate $P(n)$ to be the equation (6.1). Now the theorem can be restated as the claim that $P(n)$ is true for all $n \in \mathbb{N}$. This is great, because the induction principle lets us reach precisely that conclusion, provided we establish two simpler facts:

- $P(0)$ is true.
- For all $n \in \mathbb{N}$, $P(n)$ IMPLIES $P(n+1)$.

So now our job is reduced to proving these two statements. The first is true because $P(0)$ asserts that a sum of zero terms is equal to $0(0+1)/2 = 0$, which is true by definition.

The second statement is more complicated. But remember the basic plan from Section 1.5 for proving the validity of any implication: *assume* the statement on the left and then *prove* the statement on the right. In this case, we assume $P(n)$—namely, equation (6.1) —in order to prove $P(n+1)$, which is the equation

$$1 + 2 + 3 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2}. \tag{6.2}$$

These two equations are quite similar; in fact, adding $(n+1)$ to both sides of equation (6.1) and simplifying the right side gives the equation (6.2):

$$\begin{aligned} 1 + 2 + 3 + \cdots + n + (n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{(n+2)(n+1)}{2} \end{aligned}$$

Thus, if $P(n)$ is true, then so is $P(n+1)$. This argument is valid for every nonnegative integer n, so this establishes the second fact required by the induction principle. Therefore, the induction principle says that the predicate $P(m)$ is true for all nonnegative integers, m, so the theorem is proved.

6.1.3 A Template for Induction Proofs

The proof of equation (6.1) was relatively simple, but even the most complicated induction proof follows exactly the same template. There are five components:

1. **State that the proof uses induction.** This immediately conveys the overall structure of the proof, which helps your reader follow your argument.

2. **Define an appropriate predicate $P(n)$.** The predicate $P(n)$ is called the *induction hypothesis*. The eventual conclusion of the induction argument will be that $P(n)$ is true for all nonnegative n. Clearly stating the induction hypothesis is often the most important part of an induction proof, and omitting it is the largest source of confused proofs by students.

   In the simplest cases, the induction hypothesis can be lifted straight from the proposition you are trying to prove, as we did with equation (6.1). Sometimes the induction hypothesis will involve several variables, in which case you should indicate which variable serves as n.

3. **Prove that $P(0)$ is true.** This is usually easy, as in the example above. This part of the proof is called the *base case* or *basis step*.

4. **Prove that $P(n)$ implies $P(n+1)$ for every nonnegative integer n.** This is called the *inductive step*. The basic plan is always the same: assume that $P(n)$ is true and then use this assumption to prove that $P(n+1)$ is true. These two statements should be fairly similar, but bridging the gap may require some ingenuity. Whatever argument you give must be valid for every nonnegative integer n, since the goal is to prove the implications $P(0) \to P(1)$, $P(1) \to P(2)$, $P(2) \to P(3)$, etc. all at once.

5. **Invoke induction.** Given these facts, the induction principle allows you to conclude that $P(n)$ is true for all nonnegative n. This is the logical capstone to the whole argument, but it is so standard that it's usual not to mention it explicitly.

Always be sure to explicitly label the *base case* and the *inductive step*. It will make your proofs clearer, and it will decrease the chance that you forget a key step (such as checking the base case).

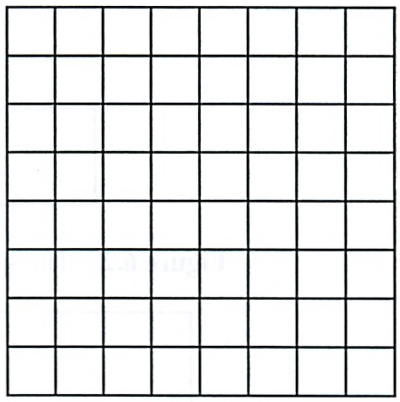
6.1.4 A Clean Writeup

The proof of the Theorem given above is perfectly valid; however, it contains a lot of extraneous explanation that you won't usually see in induction proofs. The writeup below is closer to what you might see in print and should be prepared to produce yourself.

Revised proof of the Theorem. We use induction. The induction hypothesis, $P(n)$, will be equation (6.1).

Fig_3-1.pdf

Redrawn



Wrong Place

Figure 6.1 A $2^n \times 2^n$ courtyard for $n = 3$.

Base case: $P(0)$ is true, because both sides of equation (6.1) equal zero when $n = 0$.

Inductive step: Assume that $P(n)$ is true, where n is any nonnegative integer. Then

$$1+2+3+\cdots+n+(n+1) = \frac{n(n+1)}{2} + (n+1) \quad \text{(by induction hypothesis)}$$

$$= \frac{(n+1)(n+2)}{2} \quad \text{(by simple algebra)}$$

which proves $P(n+1)$. ← how ??? -as

So it follows by induction that $P(n)$ is true for all nonnegative n. ■

Do we assume don't have to prove simple algebra? But what is the point then?

Induction was helpful for *proving the correctness* of this summation formula, but not helpful for *discovering* it in the first place. Tricks and methods for finding such formulas will be covered in Part III of the text.

6.1.5 A More Challenging Example

During the development of MIT's famous Stata Center, as costs rose further and further beyond budget, there were some radical fundraising ideas. One rumored plan was to install a big courtyard with dimensions $2^n \times 2^n$ with one of the central squares[2] occupied by a statue of a wealthy potential donor —who we will refer to

[2] In the special case $n = 0$, the whole courtyard consists of a single central square; otherwise, there are four central squares.

Fig_3-2.pdf

Redrawn

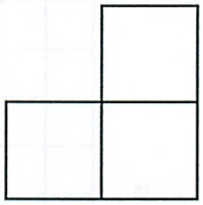


Figure 6.2 The special L-shaped tile.

Fig_3-3.pdf

Redrawn

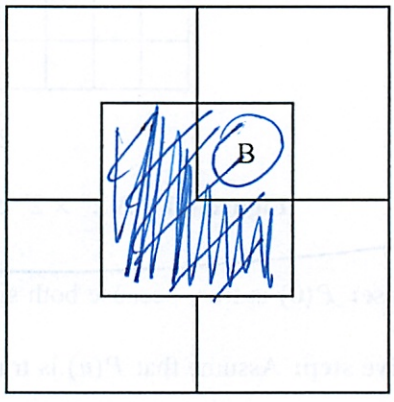


Figure 6.3 A tiling using L-shaped tiles for $n = 2$ with Bill in a center square.

as "Bill", for the purposes of preserving anonymity. The $n = 3$ case is shown in Figure 6.1.

A complication was that the building's unconventional architect, Frank Gehry, was alleged to require that only special L-shaped tiles (shown in Figure 6.2) be used for the courtyard. For $n = 2$, a courtyard meeting these constraints is shown in Figure 6.3. But what about for larger values of n? Is there a way to tile a $2^n \times 2^n$ courtyard with L-shaped tiles around a statue in the center? Let's try to prove that this is so.

Theorem 6.1.1. *For all $n \geq 0$ there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in a central square.*

Proof. *(doomed attempt)* The proof is by induction. Let $P(n)$ be the proposition that there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in the center.

Base case: $P(0)$ is true because Bill fills the whole courtyard.

Inductive step: Assume that there is a tiling of a $2^n \times 2^n$ courtyard with Bill in the center for some $n \geq 0$. We must prove that there is a way to tile a $2^{n+1} \times 2^{n+1}$ courtyard with Bill in the center .... ■

Now we're in trouble! The ability to tile a smaller courtyard with Bill in the center isn't much help in tiling a larger courtyard with Bill in the center. We haven't figured out how to bridge the gap between $P(n)$ and $P(n+1)$.

So if we're going to prove Theorem 6.1.1 by induction, we're going to need some *other* induction hypothesis than simply the statement about n that we're trying to prove.

Maybe you can figure one a good induction hypothesis for tiling. In class we'll present some hypotheses that do work.

6.1.6 A Faulty Induction Proof

If we have done a good job in writing this text, right about now you should be thinking, "Hey, this induction stuff isn't so hard after all —just show $P(0)$ is true and that $P(n)$ implies $P(n+1)$ for any number n." And, you would be right, although sometimes when you start doing induction proofs on your own, you can run into trouble. For example, we will now attempt to ruin your day by using induction to "prove" that all horses are the same color. And just when you thought it was safe to skip class and work on your robot program instead. Bummer!

False Theorem. *All horses are the same color.*

Notice that no n is mentioned in this assertion, so we're going to have to reformulate it in a way that makes an n explicit. In particular, we'll (falsely) prove that

False Theorem 6.1.2. *In every set of $n \geq 1$ horses, all the horses are the same color.*

This a statement about all integers $n \geq 1$ rather ≥ 0, so it's natural to use a slight variation on induction: prove $P(1)$ in the base case and then prove that $P(n)$ implies $P(n+1)$ for all $n \geq 1$ in the inductive step. This is a perfectly valid variant of induction and is *not* the problem with the proof below.

Bogus proof. The proof is by induction on n. The induction hypothesis, $P(n)$, will be

$$\text{In every set of } n \text{ horses, all are the same color.} \tag{6.3}$$

Base case: ($n = 1$). $P(1)$ is true, because in a set of horses of size 1, there's only one horse, and this horse is definitely the same color as itself.

Inductive step: Assume that $P(n)$ is true for some $n \geq 1$. That is, assume that in every set of n horses, all are the same color. Now suppose we have a set of $n+1$ horses:

$$h_1, h_2, \ldots, h_n, h_{n+1}.$$

We need to prove these $n+1$ horses are all the same color.

By our assumption, the first n horses are the same color:

$$\underbrace{h_1,\ h_2,\ \ldots,\ h_n}_{\text{same color}},\ h_{n+1}$$

Also by our assumption, the last n horses are the same color:

$$h_1,\ \underbrace{h_2,\ \ldots,\ h_n,\ h_{n+1}}_{\text{same color}}$$

So h_1 is the same color as the remaining horses besides h_{n+1} —that is, $h_2, \ldots, h_n$. Likewise, h_{n+1} is the same color as the remaining horses besides h_1 —that is, $h_2, \ldots, h_n$, again. Since h_1 and h_{n+1} are the same color as $h_2, \ldots, h_n$, all $n+1$ horses must be the same color, and so $P(n+1)$ is true. Thus, $P(n)$ implies $P(n+1)$.

By the principle of induction, $P(n)$ is true for all $n \geq 1$. ■

We've proved something false! Is math broken? Should we all become poets? No, this proof has a mistake.

See if you can figure it out before we take it up in class.

Students sometimes explain that the mistake in the proof is because $P(n)$ is false for $n \geq 2$, and the proof assumes something false, namely, $P(n)$, in order to prove $P(n+1)$. You should think about how to explain to such a student why this explanation would get no credit on a Math for Computer Science exam.

6.2 State Machines

State machines are a simple abstract model of step-by-step processes. Since computer programs can be understood as defining step-by-step computational processes, it's not surprising mthat state machines come up regularly in computer science. They also come up in many other settings such as digital circuit design and modeling of probabilistic processes. This section introduces *Floyd's Invariance Principle* which is a version of induction tailored specifically for proving propeorties of state machines.

One of the most important uses of induction in computer science involves proving one or more desirable properties continues to hold at every-step in a process. A property that is preserved through a series of operations or steps is known as an *invariant*. Examples of desirable invariants include properties such as a variable

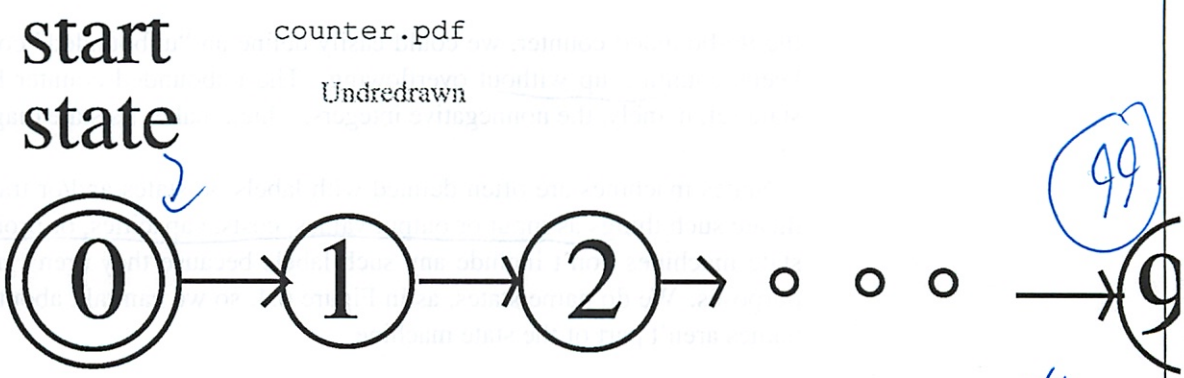


Figure 6.4 *State transitions for the 99-bounded counter.*

never exceeding a certain value, the altitude of a plane never dropping below 1,000 feet without the wingflaps being deployed, and the temperature of a nuclear reactor never exceeding the threshold for a meltdown.

6.2.1 States and Transitions

Formally, a state machine is nothing more than a binary relation on a set, except that the elements of the set are called "states," the relation is called the *transition relation*, and an arrow in the graph of the transition relation is called a *transition*. A transition from state q to state r will be written $q \longrightarrow r$. The transition relation is also called the *state graph* of the machine. A state machine also comes equipped with a designated *start state*.

A simple example is a bounded counter, which counts from 0 to 99 and overflows at 100. This state machine is pictured in Figure 6.4, with states pictured a circles, transitions by arrows, and with start state 0 indicated by the double circle. To be precise, what the picture tells us is that this bounded counter machines has

$$\begin{aligned} \text{states} &::= \{0, 1, \ldots, 99, \text{overflow}\}, \\ \text{start state} &::= 0, \\ \text{transitions} &::= \{n \longrightarrow n+1 \mid 0 \le n < 99\} \cup \{99 \longrightarrow \text{overflow}, \text{overflow} \longrightarrow \text{overflow}\}. \end{aligned}$$

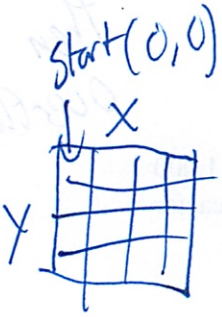
This machine isn't much use once it overflows, since it has no way to get out of its overflow state.

State machines for digital circuits and string pattern matching algorithms, for example, usually have only a finite number of states. Machines that model continuing computations typically have an infinite number of states. For example, instead of

the 99-bounded counter, we could easily define an "unbounded" counter that just keeps counting up without overflowing. The unbounded counter has an infinite state set, namely, the nonnegative integers, which makes its state diagram harder to draw. :-)

States machines are often defined with labels on states and/or transitions to indicate such things as input or output values, costs, capacities, or probabilities. Our state machines don't include any such labels because they aren't needed for our purposes. We do name states, as in Figure 6.4, so we can talk about them, but the names aren't part of the state machine.

6.2.2 Invariance for a Diagonally-Moving Robot



Suppose we have a robot that moves on an infinite 2-dimensional integer grid. The *state* of the robot at any time can be specified by the integer coordinates (x, y) of the robot's current position. The *start state* is $(0, 0)$ since it is given that the robot starts at that position. At each step, the robit may to a diagonally adjacent grid point. To be precise, robot's transitions are:

$$\{(m,n) \longrightarrow (m \pm 1, n \pm 1) \mid m,n \in \mathbb{Z}\}.$$

For example, after the first step, the robot could be in states $(1,1)$, $(1,-1)$, $(-1,1)$, or $(-1,-1)$. After two steps, there are 9 possible states for the robot, including $(0,0)$.

Can the robot ever reach position $(1,0)$?

If you play around with the robot a bit, you'll probably notice that the robot can only reach positions (m,n) for which $m+n$ is even, which means, of course, that it can't reach $(1,0)$. This all follows because evenness of the sum of coordinates is preserved by transitions.

Since can't do 0

This once, let's go through this preserved-property argument again carefully highlighting where induction comes in. Namely, define the even-sum property of states to be:

Can only move diagonally

$$\text{Even-sum}((m,n)) ::= [m+n \text{ is even}].$$

Lemma 6.2.1. *For any transition, $q \longrightarrow r$, of the diagonally-moving robot, if Even-sum(q), then Even-sum(r).*

This lemma follows immediately from the definition of the robot's transitions: $(m,n) \longrightarrow (m \pm 1, n \pm 1)$. After a transition, the sum of coordinates changes by $(\pm 1) + (\pm 1)$, that is, by 0, 2, or -2. Of course, adding 0, 2 or -2 to an even number gives an even number. So by a trivial induction on the number of transitions, we can prove:

Theorem 6.2.2. *The sum of the coordinates of any state reachable by the diagonally-moving robot is even.*

Proof. The proof is induction on the number of transitions the robot has made. The induction hypothesis is

$$P(n) ::= \text{if } q \text{ a state reachable in } n \text{ transitions, then Even-sum}(q).$$

base case: $P(0)$ is true since the only state reachable in 0 transitions is the start state $(0,0)$, and $0+0$ is even.

inductive case Assume that $P(n)$ is true, and let r be any state reachable in $n+1$ transitions. We need to prove that Even-sum(r) holds.

Now since r is reachable in $n+1$ transitions, there must be a state, q, reachable in n transitions such that $q \longrightarrow r$. Now Even-sum(q) holds since $P(n)$ is assumed to be true, so by Lemma 6.2.1, Even-sum(r) also holds. This proves that $P(n)$ IMPLIES $P(n+1)$ as required, completing the proof of the inductive step.

We conclude by induction that for all $n \geq 0$, if q is reachable in n transitions, then Even-sum(q). This implies that every reachable state has the Even-sum property. ■

Corollary 6.2.3. *The robot can never reach position* $(1,0)$.

Proof. By Theorem 6.2.2, we know the robot can only reach positions with coordinates that sum to an even number, and thus it cannot reach position $(1,0)$. ■

6.2.3 The Invariance Principle

Using the Even-sum invariant to understand the diagonally-moving robot is a simple example of a basic proof method called The Invariance Principle. The Principle summarizes how induction on the number of steps to reach a state applies to invariants. To formulate it precisely, we need a definition of *reachability*.

Definition 6.2.4. The *reachable states* of a state machine, M, are defined recursively as follows:

- the start state is reachable, and
- if p is a reachable state of M, and $p \longrightarrow q$ is a transition of M, then q is also a reachable state of M.

Definition 6.2.5. A ~~*preserved invariant*~~ of a state machine is a predicate, P, on states, such that whenever ~~$P(q)$ is true~~ of a state, q, and $q \longrightarrow r$ for some state, r, then $P(r)$ holds.

The Invariant Principle

If a preserved invariant of a state machine is true for the start state, then it is true for all reachable states.

The Invariant Principle is nothing more than the Induction Principle reformulated in a convenient form for state machines. Showing that a predicate is true in the start state is the base case of the induction, and showing that a predicate is a preserved invariant corresponds to the inductive step.[3]

[3]Preserved invariants are commonly just called "invariants" in the literature on program correctness, but we decided to throw in the extra adjective to avoid confusion with other definitions. For example, other texts (as well as another subject at MIT) use "invariant" to mean "predicate true of all reachable states." Let's call this definition "invariant-2." Now invariant-2 seems like a reasonable definition, since unreachable states by definition don't matter, and all we want to show is that a desired property is invariant-2. But this confuses the *objective* of demonstrating that a property is invariant-2 with the *method* of finding a *preserved* invariant to *show* that it is invariant-2.

I don't really get it

Robert_Floyd_pic.pdf

Undredrawn

Robert W. Floyd

The Invariant Principle was formulated by Robert Floyd at Carnegie Tech[4] in 1967. Floyd was already famous for work on formal grammars that transformed the field of programming language parsing; that was how he got to be a professor even though he never got a Ph.D. (He was admitted to a PhD program as a teenage prodigy, but flunked out and never went back.)

In that same year, Albert R Meyer was appointed Assistant Professor in the Carnegie Tech Computer Science Department where he first met Floyd. Floyd and Meyer were the only theoreticians in the department, and they were both delighted to talk about their shared interests. After just a few conversations, Floyd's new junior colleague decided that Floyd was the smartest person he had ever met.

Naturally, one of the first things Floyd wanted to tell Meyer about was his new, as yet unpublished, Invariant Principle. Floyd explained the result to Meyer, and Meyer wondered (privately) how someone as brilliant as Floyd could be excited by such a trivial observation. Floyd had to show Meyer a bunch of examples before Meyer understood Floyd's excitement —not at the truth of the utterly obvious Invariant Principle, but rather at the insight that such a simple method could be so widely and easily applied in verifying programs.

Floyd left for Stanford the following year. He won the Turing award —the "Nobel prize" of computer science— in the late 1970's, in recognition both of his work on grammars and on the foundations of program verification. He remained at Stanford from 1968 until his death in September, 2001. You can learn more about Floyd's life and work by reading the eulogy written by his closest colleague, Don Knuth.

6.2.4 The Die Hard Example

The movie *Die Hard 3: With a Vengeance* includes an amusing example of a state machine. The lead characters played by Samuel L. Jackson and Bruce Willis have to disarm a bomb planted by the diabolical Simon Gruber:

Simon: On the fountain, there should be 2 jugs, do you see them? A 5-gallon and a 3-gallon. Fill one of the jugs with exactly 4 gallons of water and place it on the scale and the timer will stop. You must be precise; one ounce more or less will result in detonation. If you're still alive in 5 minutes, we'll speak.

Bruce: Wait, wait a second. I don't get it. Do you get it?

Samuel: No.

Bruce: Get the jugs. Obviously, we can't fill the 3-gallon jug with 4 gallons of water.

Samuel: Obviously.

Bruce: All right. I know, here we go. We fill the 3-gallon jug exactly to the top, right?

Samuel: Uh-huh.

Bruce: Okay, now we pour this 3 gallons into the 5-gallon jug, giving us exactly 3 gallons in the 5-gallon jug, right?

Samuel: Right, then what?

Bruce: All right. We take the 3-gallon jug and fill it a third of the way...

Samuel: No! He said, "Be precise." Exactly 4 gallons.

Bruce: Sh - -. Every cop within 50 miles is running his a - - off and I'm out here playing kids games in the park.

Samuel: Hey, you want to focus on the problem at hand?

Fortunately, they find a solution in the nick of time. You can work out how.

The Die Hard 3 State Machine

The jug-filling scenario can be modeled with a state machine that keeps track of the amount, b, of water in the big jug, and the ammount, l, in the little jug. With the 3 and a 5 gallon water jugs, the states formally will be pairs, (b, l) of real numbers such that $0 \leq b \leq 5, 0 \leq l \leq 3$. (We can prove that the reachable values of b and l will be nonnegative integers, but we won't assume this.) The start state is $(0, 0)$, since both jugs start empty.

Since the amount of water in the jug must be known exactly, we will only consider moves in which a jug gets completely filled or completely emptied. There are several kinds of transitions:

joint sm cake

1. Fill the little jug: $(b, l) \longrightarrow (b, 3)$ for $l < 3$.
2. Fill the big jug: $(b, l) \longrightarrow (5, l)$ for $b < 5$.
3. Empty the little jug: $(b, l) \longrightarrow (b, 0)$ for $l > 0$.
4. Empty the big jug: $(b, l) \longrightarrow (0, l)$ for $b > 0$.
5. Pour from the little jug into the big jug: for $l > 0$,

$$(b, l) \longrightarrow \begin{cases} (b + l, 0) & \text{if } b + l \leq 5, \\ (5, l - (5 - b)) & \text{otherwise.} \end{cases}$$

6. Pour from big jug into little jug: for $b > 0$,

$$(b, l) \longrightarrow \begin{cases} (0, b + l) & \text{if } b + l \leq 3, \\ (b - (3 - l), 3) & \text{otherwise.} \end{cases}$$

Note that in contrast to the 99-counter state machine, there is more than one possible transition out of states in the Die Hard machine. Machines like the 99-counter with at most one transition out of each state are called *deterministic*. The Die Hard machine is *nondeterministic* because some states have transitions to several different states.

The Die Hard 3 bomb gets disarmed successfully because the state (4,3) is reachable.

Die Hard Once and For All

The *Die Hard* series is getting tired, so we propose a final *Die Hard Once and For All*. Here Simon's brother returns to avenge him, and he poses the same challenge, but with the 5 gallon jug replaced by a 9 gallon one. The state machine has the same specification as in Die Hard 3, with all occurrences of "5" replaced by "9."

Now reaching any state of the form $(4, l)$ is impossible. We prove this using the Invariant Principle. Namely, we define the preserved invariant predicate, $P((b, l))$, to be that b and l are nonnegative integer multiples of 3.

To prove that P is a preserved invariant of Die-Hard-Once-and-For-All machine, we assume $P(q)$ holds for some state $q ::= (b, l)$ and that $q \longrightarrow r$. We have to show that $P(r)$ holds. The proof divides into cases, according to which transition rule is used.

One case is a "fill the little jug" transition. This means $r = (b, 3)$. But $P(q)$ implies that b is an integer multiple of 3, and of course 3 is an integer multiple of 3, so $P(r)$ still holds.

Another case is a "pour from big jug into little jug" transition. For the subcase when there isn't enough room in the little jug to hold all the water, namely, when $b + l > 3$, we have $r = (b - (3 - l), 3)$. But $P(q)$ implies that b and l are integer multiples of 3, which means $b - (3 - l)$ is too, so in this case too, $P(r)$ holds.

We won't bother to crank out the remaining cases, which can all be checked just as easily. Now by the Invariant Principle, we conclude that every reachable state satisifies P. But since no state of the form $(4, l)$ satisifies P, we have proved rigorously that Bruce dies once and for all!

By the way, notice that the state (1,0), which satisfies NOT(P), has a transition to (0,0), which satisfies P. So the negation of a preserved invariant may not be a preserved invariant.

Read more carefully

6.2.5 Fast Exponentiation

Partial Correctness & Termination

Floyd distinguished two required properties to verify a program. The first property is called *partial correctness*; this is the property that the final results, if any, of the process must satisfy system requirements.

You might suppose that if a result was only partially correct, then it might also be partially incorrect, but that's not what Floyd meant. The word "partial" comes from viewing a process that might not terminate as computing a *partial relation*. Partial correctness means that *when there is a result*, it is correct, but the process might not always produce a result, perhaps because it gets stuck in a loop.

The second correctness property called *termination* is that the process does always produce some final value.

Partial correctness can commonly be proved using the Invariant Principle. Termination can commonly be proved using the Well Ordering Principle. We'll illustrate this by verifying a Fast Exponentiation procedure.

Exponentiating

The most straightforward way to compute the bth power of a number, a, is to multiply a by itself $b - 1$ times. There is another way to do it using considerably fewer multiplications called *Fast Exponentiation*. The register machine program below defines the fast exponentiation algorithm. The letters x, y, z, r denote registers that hold numbers. An *assignment statement* has the form "$z := a$" and has the effect of setting the number in register z to be the number a.

A Fast Exponentiation Program

Given inputs $a \in \mathbb{R}, b \in \mathbb{N}$, initialize registers x, y, z to $a, 1, b$ respectively, and repeat the following sequence of steps until termination:

- if $z = 0$ **return** y and terminate
- $r :=$ remainder$(z, 2)$
- $z :=$ quotient$(z, 2)$
- if $r = 1$, then $y := xy$
- $x := x^2$

We claim this program always terminates and leaves $y = a^b$.

To begin, we'll model the behavior of the program with a state machine:

1. states $::= \mathbb{R} \times \mathbb{R} \times \mathbb{N}$,
2. start state $::= (a, 1, b)$,
3. transitions are defined by the rule

$$(x, y, z) \longrightarrow \begin{cases} (x^2, y, \text{quotient}(z, 2)) & \text{if } z \text{ is nonzero and even,} \\ (x^2, xy, \text{quotient}(z, 2)) & \text{if } z \text{ is nonzero and odd.} \end{cases}$$

The preserved invariant, $P((x, y, z))$, will be

$$z \in \mathbb{N} \text{ AND } yx^z = a^b. \tag{6.4}$$

To prove that P is preserved, assume $P((x, y, z))$ holds and that $(x, y, z) \longrightarrow (x_t, y_t, z_t)$. We must prove that $P((x_t, y_t, z_t))$ holds, that is,

$$z_t \in \mathbb{N} \text{ AND } y_t x_t^{z_t} = a^b. \tag{6.5}$$

Since there is a transition from (x, y, z), we have $z \neq 0$, and since $z \in \mathbb{N}$ by (6.4), we can consider just two cases:

If z is even, then we have that $x_t = x^2, y_t = y, z_t = \text{quotient}(z, 2)$. Therefore, $z_t \in \mathbb{N}$ and

$$\begin{aligned} y_t x_t^{z_t} &= y x^{2 \cdot \text{quotient}(z,2)} \\ &= y x^{2 \cdot (z/2)} \\ &= y x^z \\ &= a^b \qquad \text{(by (6.4))} \end{aligned}$$

If z is odd, then we have that $x_t = x^2$, $y_t = xy$, $z_t = \text{quotient}(z, 2)$. Therefore, $z_t \in \mathbb{N}$ and

$$\begin{aligned} y_t x_t^{z_t} &= xyx^{2\cdot\text{quotient}(z,2)} \\ &= yx^{1+2\cdot(z-1)/2} \\ &= yx^{1+(z-1)} \\ &= yx^z \\ &= a^b \qquad \text{(by (6.4))} \end{aligned}$$

So in both cases, (6.5) holds, proving that P is a preserved invariant.

Now it's easy to prove partial correctness, namely, if the Fast Exponentiation program terminates, it does so with a^b in register y. This works because obviously $1 \cdot a^b = a^b$, which means that the start state, $(a, 1, b)$, satisifies P. By the Invariant Principle, P holds for all reachable states. But the program only stops when $z = 0$, so if a terminated state, $(x, y, 0)$ is reachable, then $y = yx^0 = a^b$ as required.

Ok, it's partially correct, but what's fast about it? The answer is that the number of multiplications it performs to compute a^b is roughly the length of the binary representation of b. That is, the Fast Exponentiation program uses roughly $\log_2 b$ multiplications compared to the naive approach of multiplying by a a total of $b-1$ times.

More precisely, it requires at most $2(\lceil \log_2 b \rceil + 1)$ multiplications for the Fast Exponentiation algorithm to compute a^b for $b > 1$. The reason is that the number in register z is initially b, and gets at least halved with each transition. So it can't be halved more than $\lceil \log_2 b \rceil + 1$ times before hitting zero and causing the program to terminate. Since each of the transitions involves at most two multiplications, the total number of multiplications until $z = 0$ is at most $2(\lceil \log_2 b \rceil + 1)$ for $b > 0$ (see Problem 6.25).

6.3 Strong Induction

A useful variant of induction is called *Strong Induction*. Strong induction and ordinary induction are used for exactly the same thing: proving that a predicate is true for all nonnegative integers. Strong induction is useful when a simple proof that the predicate holds for $n + 1$ does not follow just from the fact that it holds at n, but from the fact that it holds for other values $\leq n$.

6.3.1 A Rule for Strong Induction

Principle of Strong Induction.

Let P be a predicate on nonnegative integers. If

- $P(0)$ is true, and
- for all $n \in \mathbb{N}$, $P(0)$, $P(1)$, ..., $P(n)$ *together* imply $P(n+1)$,

then $P(m)$ is true for all $m \in \mathbb{N}$.

so can show true for extra values!

The only change from the ordinary induction principle is that strong induction allows you to assume more stuff in the inductive step of your proof! In an ordinary induction argument, you assume that $P(n)$ is true and try to prove that $P(n+1)$ is also true. In a strong induction argument, you may assume that $P(0)$, $P(1)$, ..., and $P(n)$ are *all* true when you go to prove $P(n+1)$. These extra assumptions can only make your job easier. Hence the name: *strong* induction.

Formulated as a proof rule, strong induction is

Rule. ***Strong Induction Rule***

$$\frac{P(0), \quad \forall n \in \mathbb{N}.\ \big(P(0) \text{ AND } P(1) \text{ AND } \ldots \text{ AND } P(n)\big) \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\ P(m)}$$

Stated more succintly, the rule is

Rule.

$$\frac{P(0), \quad [\forall k \leq n \in \mathbb{N}.\ P(k)] \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\ P(m)}$$

The template for strong induction proofs is identical to the template given in Section 6.1.3 for ordinary induction except for two things:

- you should state that your proof is by strong induction, and
- you can assume that $P(0)$, $P(1)$, ..., $P(n)$ are all true instead of only $P(n)$ during the inductive step.

6.3.2 Products of Primes

As a first example, we'll use strong induction to re-prove Theorem 2.4.1 which we previously proved using Well Ordering.

Theorem. *Every integer greater than 1 is a product of primes.*

Proof. We will prove the Theorem by strong induction, letting the induction hypothesis, $P(n)$, be

$$n \text{ is a product of primes.}$$

So the Theorem will follow if we prove that $P(n)$ holds for all $n \geq 2$.

Base Case: ($n = 2$): $P(2)$ is true because 2 is prime, so it is a length one product of primes by convention.

Inductive step: Suppose that $n \geq 2$ and that k is a product of primes for every integer k where $2 \leq k \leq n$. We must show that $P(n+1)$ holds, namely, that $n+1$ is also a product of primes. We argue by cases:

If $n+1$ is itself prime, then it is a length one product of primes by convention, and so $P(n+1)$ holds in this case.

Otherwise, $n+1$ is not prime, which by definition means $n+1 = km$ for some integers k, m such that $2 \leq k, m \leq n$. Now by the strong induction hypothesis, we know that k is a product of primes. Likewise, m is a product of primes. By multiplying these products, it follows immediately that $km = n$ is also a product of primes. Therefore, $P(n+1)$ holds in this case as well.

So $P(n+1)$ holds in any case, which completes the proof by strong induction that $P(n)$ holds for all $n \geq 2$.

■

6.3.3 Making Change

The country Inductia, whose unit of currency is the Strong, has coins worth 3Sg (3 Strongs) and 5Sg. Although the Inductians have some trouble making small change like 4Sg or 7Sg, it turns out that they can collect coins to make change for any number that is at least 8 Strongs.

Strong induction makes this easy to prove for $n+1 \geq 11$, because then $(n+1)-3 \geq 8$, so by strong induction the Inductians can make change for exactly $(n+1)-3$ Strongs, and then they can add a 3Sg coin to get $(n+1)$Sg. So the only thing to do is check that they can make change for all the amounts from 8 to 10Sg, which is not too hard to do.

Here's a detailed writeup using the official format:

Proof. We prove by strong induction that the Inductians can make change for any amount of at least 8Sg. The induction hypothesis, $P(n)$ will be:

> There is a collection of coins whose value is $n+8$ Strongs.

We now proceed with the induction proof:

Base case: $P(0)$ is true because a 3Sg coin together with a 5Sg coin makes 8Sg.

Inductive step: We assume $P(k)$ holds for all $k \leq n$, and prove that $P(n+1)$ holds. We argue by cases:

Case ($n + 1 = 1$): We have to make $(n + 1) + 8 = 9$Sg. We can do this using three 3Sg coins.

Case ($n + 1 = 2$): We have to make $(n + 1) + 8 = 10$Sg. Use two 5Sg coins.

Case ($n + 1 \geq 3$): Then $0 \leq n - 2 \leq n$, so by the strong induction hypothesis, the Inductians can make change for $n - 2$ Strong. Now by adding a 3Sg coin, they can make change for $(n + 1)$Sg.

Since $n \geq 0$, we know that $n + 1 \geq 1$ and thus that the three cases cover every possibility. Since $P(n + 1)$ is true in every case, we can conclude by strong induction that for all $n \geq 0$, the Inductians can make change for $n + 8$ Strong. That is, they can make change for any number of eight or more Strong. ■

6.3.4 The Stacking Game

Here is another exciting game that's surely about to sweep the nation!

You begin with a stack of n boxes. Then you make a sequence of moves. In each move, you divide one stack of boxes into two nonempty stacks. The game ends when you have n stacks, each containing a single box. You earn points for each move; in particular, if you divide one stack of height $a + b$ into two stacks with heights a and b, then you score ab points for that move. Your overall score is the sum of the points that you earn for each move. What strategy should you use to maximize your total score?

As an example, suppose that we begin with a stack of $n = 10$ boxes. Then the game might proceed as shown in Figure 6.5. Can you find a better strategy?

Analyzing the Game

You will see in class how to use strong induction to analyze this game of blocks.

6.4 Strong Induction vs. Induction vs. Well Ordering

Strong induction looks genuinely "stronger" than ordinary induction —after all, you can assume a lot more when proving the induction step. Since ordinary induction is a special case of strong induction, you might wonder why anyone would bother with the ordinary induction.

But strong induction really isn't any stronger, because a simple text manipulation program can automatically reformat any proof using strong induction into a proof using ordinary induction —just by decorating the induction hypothesis with

Stack Heights										Score
$\underline{10}$										
5	$\underline{5}$									25 points
$\underline{5}$	3	2								6
$\underline{4}$	3	2	1							4
2	$\underline{3}$	2	1	2						4
$\underline{2}$	2	2	1	2	1					2
1	$\underline{2}$	2	1	2	1	1				1
1	1	$\underline{2}$	1	2	1	1	1			1
1	1	1	1	$\underline{2}$	1	1	1	1		1
1	1	1	1	1	1	1	1	1	1	1
								Total Score	=	45 points

Figure 6.5 An example of the stacking game with $n = 10$ boxes. On each line, the underlined stack is divided in the next step.

a universal quantifier in a standard way. Still, it's worth distinguishing these two kinds of induction, since which you use will signal whether the inductive step for $n+1$ follows directly from the case for n or requires cases smaller than n, and that is generally good for your reader to know.

The template for the two kinds of induction rules look nothing like the one for theWell Ordering Principle, but this chapter included a couple of examples where induction was used to prove something already proved using Well Ordering. In fact, this can always be done. As the examples may suggest, any Well Ordering proof can automatically be reformatted it into an Induction proof. So theoretically, no one need bother with the Well Ordering Principle either.

But wait a minute —it's equally easy go the other way, and automatically reformat any Strong Induction proof into a Well Ordering proof. The three proof methods –Well Ordering, Induction, and Strong Induction, are simply different formats for presenting the same mathematical reasoning!

So why three methods? Well, sometimes induction proofs are clearer because they don't require proof by contradiction. Also, induction proofs often provide recursive procedures that reduce handling large inputs to handling smaller ones. On the other hand, Well Ordering come out slightly shorter and sometimes seem more natural (and less worrisome to beginners).

So which method should you use? —whichever you find easier! But whichever method you choose, be sure to state the method up front to help a reader follow your proof.

Problems for Section 6.1

Class Problems

Problem 6.1.
Use induction to prove that

$$1^3 + 2^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2. \tag{6.6}$$

for all $n \geq 1$.

Remember to formally

1. Declare proof by induction.
2. Identify the induction hypothesis $P(n)$.
3. Establish the base case.
4. Prove that $P(n) \Rightarrow P(n+1)$.
5. Conclude that $P(n)$ holds for all $n \geq 1$.

as in the five part template.

Problem 6.2.
Prove by induction on n that

$$1 + r + r^2 + \cdots + r^n = \frac{r^{n+1} - 1}{r - 1} \tag{6.7}$$

for all $n \in \mathbb{N}$ and numbers $r \neq 1$.

Problem 6.3.
Prove by induction:

$$1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} < 2 - \frac{1}{n}, \tag{6.8}$$

for all $n > 1$.

Problem 6.4. **(a)** Prove by induction that a $2^n \times 2^n$ courtyard with a 1×1 statue of Bill in *a corner* can be covered with L-shaped tiles. (Do not assume or reprove the (stronger) result of Theorem 6.1.1 that Bill can be placed anywhere. The point of this problem is to show a different induction hypothesis that works.)

(b) Use the result of part (a) to prove the original claim that there is a tiling with Bill in the middle.

Problem 6.5.
Find the flaw in the following bogus proof that $a^n = 1$ for all nonnegative integers n, whenever a is a nonzero real number.

Bogus proof. The proof is by induction on n, with hypothesis

$$P(n) ::= \forall k \leq n. a^k = 1,$$

where k is a nonnegative integer valued variable.

Base Case: $P(0)$ is equivalent to $a^0 = 1$, which is true by definition of a^0. (By convention, this holds even if $a = 0$.)

Inductive Step: By induction hypothesis, $a^k = 1$ for all $k \in \mathbb{N}$ such that $k \leq n$. But then

$$a^{n+1} = \frac{a^n \cdot a^n}{a^{n-1}} = \frac{1 \cdot 1}{1} = 1,$$

which implies that $P(n+1)$ holds. It follows by induction that $P(n)$ holds for all $n \in \mathbb{N}$, and in particular, $a^n = 1$ holds for all $n \in \mathbb{N}$.

■

Problem 6.6.
We've proved in two different ways that

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

But now we're going to prove a *contradictory* theorem!

False Theorem. *For all* $n \geq 0$,

$$2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2}$$

Proof. We use induction. Let $P(n)$ be the proposition that $2+3+4+\cdots+n = n(n+1)/2$.

Base case: $P(0)$ is true, since both sides of the equation are equal to zero. (Recall that a sum with no terms is zero.)

Inductive step: Now we must show that $P(n)$ implies $P(n+1)$ for all $n \geq 0$. So suppose that $P(n)$ is true; that is, $2+3+4+\cdots+n = n(n+1)/2$. Then we can reason as follows:

$$\begin{aligned} 2+3+4+\cdots+n+(n+1) &= [2+3+4+\cdots+n]+(n+1) \\ &= \frac{n(n+1)}{2}+(n+1) \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

Above, we group some terms, use the assumption $P(n)$, and then simplify. This shows that $P(n)$ implies $P(n+1)$. By the principle of induction, $P(n)$ is true for all $n \in \mathbb{N}$. ■

Where exactly is the error in this proof?

Homework Problems

Problem 6.7.

Claim 6.4.1. *If a collection of positive integers (not necessarily distinct) has sum $n \geq 1$, then the collection has product at most $3^{n/3}$.*

For example, the collection 2, 2, 3, 4, 4, 7 has the sum:

$$2+2+3+4+4+7 = 22$$

On the other hand, the product is:

$$\begin{aligned} 2\cdot 2\cdot 3\cdot 4\cdot 4\cdot 7 &= 1344 \\ &\leq 3^{22/3} \\ &\approx 3154.2 \end{aligned}$$

(a) Use strong induction to prove that $n \leq 3^{n/3}$ for every integer $n \geq 0$.

(b) Prove the claim using induction or strong induction. (You may find it easier to use induction on the *number of positive integers in the collection* rather than induction on the sum n.)

Problem 6.8.
For any binary string, α, let $\text{num}(\alpha)$ be the nonnegative integer it represents in binary notation. For example, $\text{num}(10) = 2$, and $\text{num}(0101) = 5$.

An $n+1$-*bit adder* adds two $n+1$-bit binary numbers. More precisely, an $n+1$-bit adder takes two length $n+1$ binary strings

$$\alpha_n ::= a_n \ldots a_1 a_0,$$
$$\beta_n ::= b_n \ldots b_1 b_0,$$

and a binary digit, c_0, as inputs, and produces a length $n+1$ binary string

$$\sigma_n ::= s_n \ldots s_1 s_0,$$

and a binary digit, c_{n+1}, as outputs, and satisfies the specification:

$$\text{num}(\alpha_n) + \text{num}(\beta_n) + c_0 = 2^{n+1} c_{n+1} + \text{num}(\sigma_n). \tag{6.9}$$

There is a straighforward way to implement an $n+1$-bit adder as a digital circuit: an $n+1$-bit *ripple-carry circuit* has $1 + 2(n+1)$ binary inputs

$$a_n, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0, c_0,$$

and $n+2$ binary outputs,

$$c_{n+1}, s_n, \ldots, s_1, s_0.$$

As in Problem 3.5, the ripple-carry circuit is specified by the following formulas:

$$s_i ::= a_i \text{ XOR } b_i \text{ XOR } c_i \tag{6.10}$$
$$c_{i+1} ::= (a_i \text{ AND } b_i) \text{ OR } (a_i \text{ AND } c_i) \text{ OR } (b_i \text{ AND } c_i),. \tag{6.11}$$

for $0 \leq i \leq n$.

(a) Verify that definitions (6.10) and (6.11) imply that

$$a_n + b_n + c_n = 2c_{n+1} + s_n. \tag{6.12}$$

for all $n \in \mathbb{N}$.

(b) Prove by induction on n that an $n+1$-bit ripple-carry circuit really is an $n+1$-bit adder, that is, its outputs satisfy (6.9).

Hint: You may assume that, by definition of binary representation of integers,

$$\text{num}(\alpha_{n+1}) = a_{n+1} 2^{n+1} + \text{num}(\alpha_n). \tag{6.13}$$

Problem 6.9.

The Math for Computer Science mascot, Theory Hippotamus, made a startling discovery while playing with his prized collection of unit squares over the weekend. Here is what happened.

First, Theory Hippotamus put his favorite unit square down on the floor as in Figure 6.6 (a). He noted that the length of the periphery of the resulting shape was 4, an even number. Next, he put a second unit square down next to the first so that the two squares shared an edge as in Figure 6.6 (b). He noticed that the length of the periphery of the resulting shape was now 6, which is also an even number. (The periphery of each shape in the figure is indicated by a thicker line.) Theory Hippotamus continued to place squares so that each new square shared an edge with at least one previously-placed square and no squares overlapped. Eventually, he arrived at the shape in Figure 6.6 (c). He realized that the length of the periphery of this shape was 36, which is again an even number.

Our plucky porcine pal is perplexed by this peculiar pattern. Use induction on the number of squares to prove that the length of the periphery is always even, no matter how many squares Theory Hippotamus places or how he arranges them.

f98ps2-c.pdf

Redrawn

f98ps2-b.pdf

Redrawn

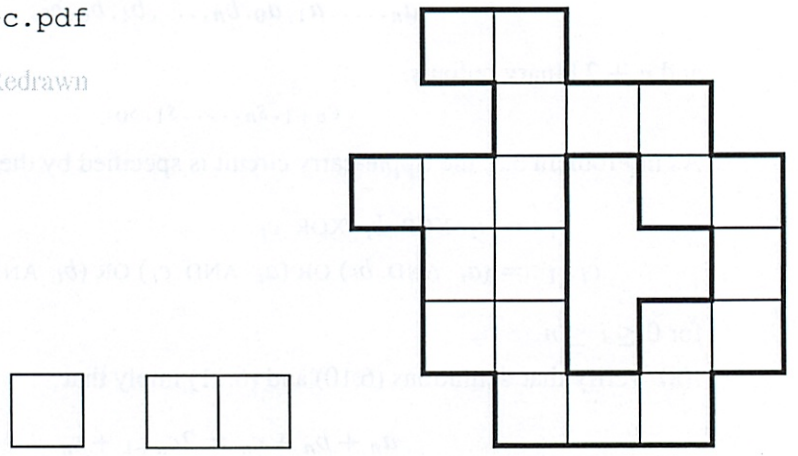


Figure 6.6 Some shapes that Theory Hippotamus created.

Problems for Section 6.2

Practice Problems

Problem 6.10.

Which states of the Die Hard 3 machine below have transitions to exactly two

states?

Die Hard Transitions

1. Fill the little jug: $(b,l) \longrightarrow (b,3)$ for $l < 3$.
2. Fill the big jug: $(b,l) \longrightarrow (5,l)$ for $b < 5$.
3. Empty the little jug: $(b,l) \longrightarrow (b,0)$ for $l > 0$.
4. Empty the big jug: $(b,l) \longrightarrow (0,l)$ for $b > 0$.
5. Pour from the little jug into the big jug: for $l > 0$,
$$(b,l) \longrightarrow \begin{cases} (b+l,0) & \text{if } b+l \leq 5, \\ (5,l-(5-b)) & \text{otherwise.} \end{cases}$$
6. Pour from big jug into little jug: for $b > 0$,
$$(b,l) \longrightarrow \begin{cases} (0,b+l) & \text{if } b+l \leq 3, \\ (b-(3-l),3) & \text{otherwise.} \end{cases}$$

Problems for Section 6.2

Homework Problems

Problem 6.11.
You are given two buckets, A and B, a water hose, a receptacle, and a drain. The buckets and receptacle are initially empty. The buckets are labeled with their respectively capacities, positive integers a and b. The receptacle can be used to store an unlimited amount of water, but has no measurement markings. Excess water can be dumped into the drain. Among the possible moves are:

1. fill a bucket from the hose,
2. pour from the receptacle to a bucket until the bucket is full or the receptacle is empty, whichever happens first,
3. empty a bucket to the drain,
4. empty a bucket to the receptacle,
5. pour from A to B until either A is empty or B is full, whichever happens first,

6. pour from B to A until either B is empty or A is full, whichever happens first.

(a) Model this scenario with a state machine. (What are the states? How does a state change in response to a move?)

(b) Prove that we can put $k \in \mathbb{N}$ gallons of water into the receptacle using the above operations if and only if $\gcd(a,b) \mid k$. *Hint:* Use the fact that if a, b are positive integers then there exist integers s, t such that $\gcd(a,b) = sa + tb$ from Section **??**.

Problem 6.12.

Here is a *very, very fun* game. We start with two distinct, positive integers written on a blackboard. Call them a and b. You and I now take turns. (I'll let you decide who goes first.) On each player's turn, he or she must write a new positive integer on the board that is the difference of two numbers that are already there. If a player can not play, then he or she loses.

For example, suppose that 12 and 15 are on the board initially. Your first play must be 3, which is $15 - 12$. Then I might play 9, which is $12 - 3$. Then you might play 6, which is $15 - 9$. Then I can not play, so I lose.

(a) Show that every number on the board at the end of the game is a multiple of $\gcd(a,b)$.

(b) Show that every positive multiple of $\gcd(a,b)$ up to $\max(a,b)$ is on the board at the end of the game.

(c) Describe a strategy that lets you win this game every time.

Problem 6.13.

In the late 1960s, the military junta that ousted the government of the small republic of Nerdia completely outlawed built-in multiplication operations, and also forbade division by any number other than 3. Fortunately, a young dissident found a way to help the population multiply any two nonnegative integers without risking persecution by the junta. The procedure he taught people is:

procedure *multiply*(x, y: nonnegative integers)
$r := x$;
$s := y$;
$a := 0$;

```
while s ≠ 0 do
    if 3 | s then
        r := r + r + r;
        s := s/3;
    else if 3 | (s − 1) then
        a := a + r;
        r := r + r + r;
        s := (s − 1)/3;
    else
        a := a + r + r;
        r := r + r + r;
        s := (s − 2)/3;
return a;
```

We can model the algorithm as a state machine whose states are triples of nonnegative integers (r, s, a). The initial state is $(x, y, 0)$. The transitions are given by the rule that for $s > 0$:

$$(r, s, a) \to \begin{cases} (3r, s/3, a) & \text{if } 3 \mid s \\ (3r, (s-1)/3, a+r) & \text{if } 3 \mid (s-1) \\ (3r, (s-2)/3, a+2r) & \text{otherwise.} \end{cases}$$

(a) List the sequence of steps that appears in the execution of the algorithm for inputs $x = 5$ and $y = 10$.

(b) Use the Invariant Method to prove that the algorithm is partially correct—that is, if $s = 0$, then $a = xy$.

(c) Prove that the algorithm terminates after at most $1 + \log_3 y$ executions of the body of the `do` statement.

Problem 6.14.

A robot named Wall-E wanders around a two-dimensional grid. He starts out at $(0, 0)$ and is allowed to take four different types of step:

1. $(+2, -1)$
2. $(+1, -2)$
3. $(+1, +1)$

4. $(-3, 0)$

Thus, for example, Wall-E might walk as follows. The types of his steps are listed above the arrows.

$$(0,0) \xrightarrow{1} (2,-1) \xrightarrow{3} (3,0) \xrightarrow{2} (4,-2) \xrightarrow{4} (1,-2) \to \ldots$$

Wall-E's true love, the fashionable and high-powered robot, Eve, awaits at $(0, 2)$.

(a) Describe a state machine model of this problem.

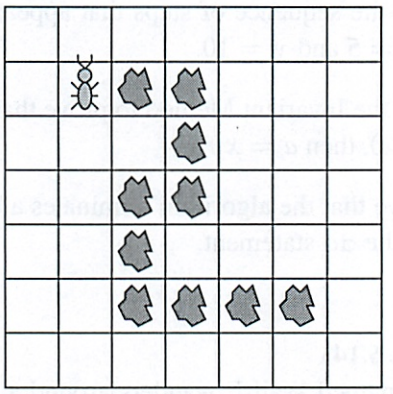
(b) Will Wall-E ever find his true love? Either find a path from Wall-E to Eve or use the Invariant Principle to prove that no such path exists.

Problem 6.15.

A hungry ant is placed on an unbounded grid. Each square of the grid either contains a crumb or is empty. The squares containing crumbs form a path in which, except at the ends, every crumb is adjacent to exactly two other crumbs. The ant is placed at one end of the path and on a square containing a crumb. For example, the figure below shows a situation in which the ant faces North, and there is a trail of food leading approximately Southeast. The ant has already eaten the crumb upon which it was initially placed.

`ant.pdf`

Undredrawn



The ant can only smell food directly in front of it. The ant can only remember a small number of things, and what it remembers after any move only depends on what it remembered and smelled immediately before the move. Based on smell and memory, the ant may choose to move forward one square, or it may turn right or left. It eats a crumb when it lands on it.

The above scenario can be nicely modelled as a state machine in which each state is a pair consisting of the "ant's memory" and "everything else"—for example, information about where things are on the grid. Work out the details of such a model state machine; design the ant-memory part of the state machine so the ant will eat all the crumbs on *any* finite path at which it starts and then signal when it is done. Be sure to clearly describe the possible states, transitions, and inputs and outputs (if any) in your model. Briefly explain why your ant will eat all the crumbs.

Note that the last transition is a self-loop; the ant signals done for eternity. One could also add another end state so that the ant signals done only once.

Problem 6.16.

Suppose that you have a regular deck of cards arranged as follows, from top to bottom:

$$A\heartsuit\ 2\heartsuit \ldots K\heartsuit\ A\spadesuit\ 2\spadesuit \ldots K\spadesuit\ A\clubsuit\ 2\clubsuit \ldots K\clubsuit\ A\diamondsuit\ 2\diamondsuit \ldots K\diamondsuit$$

Only two operations on the deck are allowed: *inshuffling* and *outshuffling*. In both, you begin by cutting the deck exactly in half, taking the top half into your right hand and the bottom into your left. Then you shuffle the two halves together so that the cards are perfectly interlaced; that is, the shuffled deck consists of one card from the left, one from the right, one from the left, one from the right, etc. The top card in the shuffled deck comes from the right hand in an outshuffle and from the left hand in an inshuffle.

(a) Model this problem as a state machine.

(b) Use the Invariant Principle to prove that you can not make the entire first half of the deck black through a sequence of inshuffles and outshuffles.

Note: Discovering a suitable invariant can be difficult! The standard approach is to identify a bunch of reachable states and then look for a pattern, some feature that they all share.[5]

Class Problems

Problem 6.17.

In this problem you will establish a basic property of a puzzle toy called the *Fifteen Puzzle* using the method of invariants. The Fifteen Puzzle consists of sliding square tiles numbered $1, \ldots, 15$ held in a 4×4 frame with one empty square. Any tile adjacent to the empty square can slide into it.

[5] If this does not work, consider twitching and drooling until someone takes the problem away.

The standard initial position is

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

We would like to reach the target position (known in the oldest author's youth as "the impossible"):

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	

A state machine model of the puzzle has states consisting of a 4×4 matrix with 16 entries consisting of the integers $1, \ldots, 15$ as well as one "empty" entry—like each of the two arrays above.

The state transitions correspond to exchanging the empty square and an adjacent numbered tile. For example, an empty at position $(2, 2)$ can exchange position with tile above it, namely, at position $(1, 2)$:

n_1	n_2	n_3	n_4
n_5		n_6	n_7
n_8	n_9	n_{10}	n_{11}
n_{12}	n_{13}	n_{14}	n_{15}

$\longrightarrow$

n_1		n_3	n_4
n_5	n_2	n_6	n_7
n_8	n_9	n_{10}	n_{11}
n_{12}	n_{13}	n_{14}	n_{15}

We will use the invariant method to prove that there is no way to reach the target state starting from the initial state.

We begin by noting that a state can also be represented as a pair consisting of two things:

1. a list of the numbers $1, \ldots, 15$ in the order in which they appear—reading rows left-to-right from the top row down, ignoring the empty square, and

2. the coordinates of the empty square—where the upper left square has coordinates $(1, 1)$, the lower right $(4, 4)$.

(a) Write out the "list" representation of the start state and the "impossible" state.

Let L be a list of the numbers $1, \ldots, 15$ in some order. A pair of integers is an *out-of-order pair* in L when the first element of the pair both comes *earlier* in the list and *is larger*, than the second element of the pair. For example, the list $1, 2, 4, 5, 3$ has two out-of-order pairs: (4,3) and (5,3). The increasing list $1, 2 \ldots n$ has no out-of-order pairs.

Let a state, S, be a pair $(L, (i, j))$ described above. We define the *parity* of S to be the mod 2 sum of the number, $p(L)$, of out-of-order pairs in L and the row-number of the empty square, that is the parity of S is $p(L) + i \pmod 2$.

(b) Verify that the parity of the start state and the target state are different.

(c) Show that the parity of a state is preserved under transitions. Conclude that "the impossible" is impossible to reach.

By the way, if two states have the same parity, then in fact there *is* a way to get from one to the other. If you like puzzles, you'll enjoy working this out on your own.

Problem 6.18.

A robot moves on the two-dimensional integer grid. It starts out at $(0, 0)$, and is allowed to move in any of these four ways:

1. (+2,-1) Right 2, down 1
2. (-2,+1) Left 2, up 1
3. (+1,+3)
4. (-1,-3)

Prove that this robot can never reach (1,1).

Problem 6.19.

The Massachusetts Turnpike Authority is concerned about the integrity of the new Zakim bridge. Their consulting architect has warned that the bridge may collapse if more than 1000 cars are on it at the same time. The Authority has also been warned by their traffic consultants that the rate of accidents from cars speeding across bridges has been increasing.

Both to lighten traffic and to discourage speeding, the Authority has decided to make the bridge *one-way* and to put tolls at *both* ends of the bridge (don't laugh, this is Massachusetts). So cars will pay tolls both on entering and exiting the bridge, but the tolls will be different. In particular, a car will pay \$3 to enter onto the bridge and will pay \$2 to exit. To be sure that there are never too many cars on the bridge, the Authority will let a car onto the bridge only if the difference between the amount of money currently at the entry toll booth minus the amount at the exit toll booth is strictly less than a certain threshold amount of $\$T_0$.

The consultants have decided to model this scenario with a state machine whose states are triples of natural numbers, (A, B, C), where

- A is an amount of money at the entry booth,
- B is an amount of money at the exit booth, and
- C is a number of cars on the bridge.

Any state with $C > 1000$ is called a *collapsed* state, which the Authority dearly hopes to avoid. There will be no transition out of a collapsed state.

Since the toll booth collectors may need to start off with some amount of money in order to make change, and there may also be some number of "official" cars already on the bridge when it is opened to the public, the consultants must be ready to analyze the system started at *any* uncollapsed state. So let A_0 be the initial number of dollars at the entrance toll booth, B_0 the initial number of dollars at the exit toll booth, and $C_0 \leq 1000$ the number of official cars on the bridge when it is opened. You should assume that even official cars pay tolls on exiting or entering the bridge after the bridge is opened.

(a) Give a mathematical model of the Authority's system for letting cars on and off the bridge by specifying a transition relation between states of the form (A, B, C) above.

The Authority has asked their engineering consultants to determine T and to verify that this policy will keep the number of cars from exceeding 1000.

The consultants reason that if C_0 is the number of official cars on the bridge when it is opened, then an additional $1000 - C_0$ cars can be allowed on the bridge. So as long as $A - B$ has not increased by $3(1000 - C_0)$, there shouldn't more than 1000 cars on the bridge. So they recommend defining

$$T_0 ::= 3(1000 - C_0) + (A_0 - B_0), \tag{6.14}$$

where A_0 is the initial number of dollars at the entrance toll booth, B_0 is the initial number of dollars at the exit toll booth.

(b) Let $D_0 ::= 2A_0 - 3B_0 - 6C_0$ and define

$$P(A, B, C) ::= [2A - 3B - 6C = D_0] \text{ AND } [C \leq 1000].$$

Verify that P is a preserved invariant of the state machine.

(c) Conclude that the traffic won't cause the bridge to collapse.

(d) A clever MIT intern working for the Turnpike Authority agrees that the Turnpike's bridge management policy will be *safe*: the bridge will not collapse. But she warns her boss that the policy will lead to *deadlock*—a situation where traffic can't move on the bridge even though the bridge has not collapsed.

Explain more precisely in terms of system transitions what the intern means, and briefly, but clearly, justify her claim.

Problem 6.20.

Start with 102 coins on a table, 98 showing heads and 4 showing tails. There are two ways to change the coins:

(i) flip over any ten coins, or

(ii) let n be the number of heads showing. Place $n + 1$ additional coins, all showing tails, on the table.

For example, you might begin by flipping nine heads and one tail, yielding 90 heads and 12 tails, then add 91 tails, yielding 90 heads and 103 tails.

(a) Model this situation as a state machine, carefully defining the set of states, the start state, and the possible state transitions.

(b) Explain how to reach a state with exactly one tail showing.

(c) Define the following derived variables:

C	::=	the number of coins on the table,	H	::=	the number of heads,
T	::=	the number of tails,	C_2	::=	remainder$(C/2)$,
H_2	::=	remainder$(H/2)$,	T_2	::=	remainder$(T/2)$.

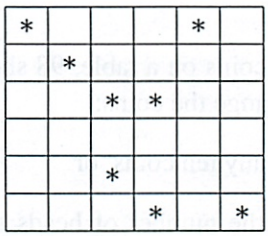
Which of these variables is

1. strictly increasing
2. weakly increasing
3. strictly decreasing
4. weakly decreasing
5. constant

(d) Prove that it is not possible to reach a state in which there is exactly one head showing.

Problem 6.21.
A classroom is designed so students sit in a square arrangement. An outbreak of beaver flu sometimes infects students in the class; beaver flu is a rare variant of bird flu that lasts forever, with symptoms including a yearning for more quizzes and the thrill of late night problem set sessions.

Here is an illustration of a 6×6-seat classroom with seats represented by squares. The locations of infected students are marked with an asterisk.

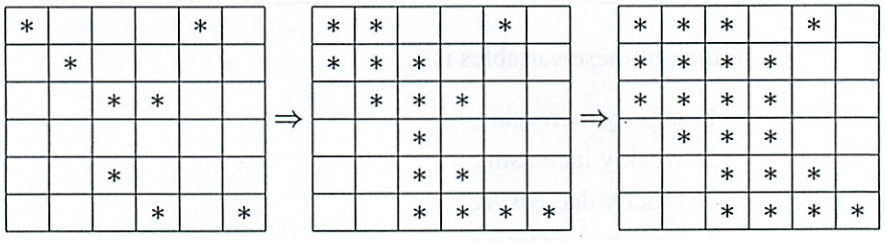


Outbreaks of infection spread rapidly step by step. A student is infected after a step if either

- the student was infected at the previous step (since beaver flu lasts forever), or
- the student was adjacent to *at least two* already-infected students at the previous step.

Here *adjacent* means the students' individual squares share an edge (front, back, left or right); they are not adjacent if they only share a corner point. So each student is adjacent to 2, 3 or 4 others.

In the example, the infection spreads as shown below.



In this example, over the next few time-steps, all the students in class become infected.

Theorem. *If fewer than n students among those in an $n\times n$ arrangment are initially infected in a flu outbreak, then there will be at least one student who never gets infected in this outbreak, even if students attend all the lectures.*

Prove this theorem.

Hint: Think of the state of an outbreak as an $n \times n$ square above, with asterisks indicating infection. The rules for the spread of infection then define the transitions of a state machine. Show that

$$R(q)::=\text{The "perimeter"'of the "infected region"}$$
$$\text{of state } q \text{ is at most } k,$$

is a preserved invariant.

Problems for Section 6.3

Class Problems

Problem 6.22.

A sequence of numbers is *weakly decreasing* when each number in the sequence is $\geq$ the numbers after it. (This implies that a sequence of just one number is weakly decreasing.)

Here's a bogus proof of a very important true fact, every integer 1 is a product of a unique weakly decreasing sequence of primes —a pusp, for short.

Explain what's bogus about the proof.

Lemma 6.4.2. *Every integer greater than 1 is a pusp.*

For example, $252 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 7$

Bogus proof. We will prove Lemma 6.4.2 by strong induction, letting the induction hypothesis, $P(n)$, be

$$n \text{ is a pusp.}$$

So Lemma 6.4.2 will follow if we prove that $P(n)$ holds for all $n \geq 2$.

Base Case: ($n = 2$) $P(2)$ is true because 2 is prime, and so it is a length one product of primes, and this is obviously the only sequence of primes whose product can equal 2.

Inductive step: Suppose that $n \geq 2$ and that i is a pusp for every integer i where $2 \leq i < n + 1$. We must show that $P(n + 1)$ holds, namely, that $n + 1$ is also a pusp. We argue by cases:

If $n + 1$ is itself prime, then it is the product of a length one sequence consisting of itself. This sequence is unique, since by definition of prime, $n + 1$ has no other prime factors. So $n + 1$ is a pusp, that is $P(n + 1)$ holds in this case.

Otherwise, $n + 1$ is not prime, which by definition means $n + 1 = km$ for some integers k, m such that $2 \leq k, m < n + 1$. Now by the strong induction hypothesis, we know that k and m are pusps. It follows immediately that by merging the unique

prime sequences for k and m, in sorted order, we get a unique weakly decreasing sequence of primes whose product equals $n+1$. So $n+1$ is a pusp, in this case as well.

So $P(n+1)$ holds in any case, which completes the proof by strong induction that $P(n)$ holds for all $n \geq 2$.

■

Problem 6.23.

Define the *potential*, $p(S)$, of a stack of blocks, S, to be $k(k-1)/2$ where k is the number of blocks in S. Define the potential, $p(A)$, of a set of stacks, A, to be the sum of the potentials of the stacks in A.

Generalize Theorem **??** about scores in the stacking game to show that for any set of stacks, A, if a sequence of moves starting with A leads to another set of stacks, B, then $p(A) \geq p(B)$, and the score for this sequence of moves is $p(A) - p(B)$.

Hint: Try induction on the number of moves to get from A to B.

Homework Problems

Problem 6.24.

A group of $n \geq 1$ people can be divided into teams, each containing either 4 or 7 people. What are all the possible values of n? Use induction to prove that your answer is correct.

Problem 6.25.

Prove that the fast exponentiation state machine of Section 6.2.5 will halt after

$$\lceil \log_2 n \rceil + 1 \tag{6.15}$$

transitions starting from any state where the value of z is $n \in integers^+$.

Hint: Strong induction.

Problem 6.26.

The following Lemma is true, but the *proof* given for it below is defective. Pinpoint *exactly* where the proof first makes an unjustified step and explain why it is unjustified.

Lemma 6.4.3. *For any prime p and positive integers $n, x_1, x_2, \ldots, x_n$, if $p \mid x_1 x_2 \ldots x_n$, then $p \mid x_i$ for some $1 \leq i \leq n$.*

early printout

Has the corrections

7 Induction

Induction is by far the most powerful and commonly-used proof technique in discrete mathematics and computer science. In fact, the use of induction is a defining characteristic of *discrete* —as opposed to *continuous* —mathematics. To understand how it works, suppose there is a professor who brings to class a bottomless bag of assorted miniature candy bars. She offers to share the candy in the following way. First, she lines the students up in order. Next she states two rules:

1. The student at the beginning of the line gets a candy bar.
2. If a student gets a candy bar, then the following student in line also gets a candy bar.

good explanation!

Let's number the students by their order in line, starting the count with 0, as usual in Computer Science. Now we can understand the second rule as a short description of a whole sequence of statements:

- If student 0 gets a candy bar, then student 1 also gets one.
- If student 1 gets a candy bar, then student 2 also gets one.
- If student 2 gets a candy bar, then student 3 also gets one.

$\vdots$

Of course this sequence has a more concise mathematical description:

> If student n gets a candy bar, then student $n + 1$ gets a candy bar, for all nonnegative integers n.

So suppose you are student 17. By these rules, are you entitled to a miniature candy bar? Well, student 0 gets a candy bar by the first rule. Therefore, by the second rule, student 1 also gets one, which means student 2 gets one, which means student 3 gets one as well, and so on. By 17 applications of the professor's second rule, you get your candy bar! Of course the rules actually guarantee a candy bar to *every* student, no matter how far back in line they may be.

7.1 Ordinary Induction

The reasoning that led us to conclude every student gets a candy bar is essentially all there is to induction.

The Principle of Induction.
Let $P(n)$ be a predicate. If

- $P(0)$ is true, and
- $P(n)$ IMPLIES $P(n+1)$ for all nonnegative integers, n,

then

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

Since we're going to consider several useful variants of induction in later sections, we'll refer to the induction method described above as *ordinary induction* when we need to distinguish it. Formulated as a proof rule, this would be

Rule. ***Induction Rule***

$$\frac{P(0), \quad \forall n \in \mathbb{N}\,[P(n) \text{ IMPLIES } P(n+1)]}{\forall m \in \mathbb{N}.\, P(m)}$$

This general induction rule works for the same intuitive reason that all the students get candy bars, and we hope the explanation using candy bars makes it clear why the soundness of the ordinary induction can be taken for granted. In fact, the rule is so obvious that it's hard to see what more basic principle could be used to justify it.[1] What's not so obvious is how much mileage we get by using it.

7.1.1 Using Ordinary Induction

Ordinary induction often works directly in proving that some statement about nonnegative integers holds for all of them. For example, here is the formula for the sum of the nonnegative integer that we already proved (equation (2.1)) using the Well Ordering Principle:

Theorem 7.1.1. *For all $n \in \mathbb{N}$,*

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \tag{7.1}$$

[1] But see section 7.3.

This time, let's use the Induction Principle to prove Theorem 7.1.1.

Suppose that we define predicate $P(n)$ to be the equation (7.1). Recast in terms of this predicate, the theorem claims that $P(n)$ is true for all $n \in \mathbb{N}$. This is great, because the induction principle lets us reach precisely that conclusion, provided we establish two simpler facts:

- $P(0)$ is true.
- For all $n \in \mathbb{N}$, $P(n)$ IMPLIES $P(n+1)$.

So now our job is reduced to proving these two statements. The first is true because $P(0)$ asserts that a sum of zero terms is equal to $0(0+1)/2 = 0$, which is true by definition. The second statement is more complicated. But remember the basic plan for proving the validity of any implication: *assume* the statement on the left and then *prove* the statement on the right. In this case, we assume $P(n)$ in order to prove $P(n+1)$, which is the equation

$$1+2+3+\cdots+n+(n+1) = \frac{(n+1)(n+2)}{2}. \tag{7.2}$$

These two equations are quite similar; in fact, adding $(n+1)$ to both sides of equation (7.1) and simplifying the right side gives the equation (7.2):

$$\begin{aligned} 1+2+3+\cdots+n+(n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{(n+2)(n+1)}{2} \end{aligned}$$

Thus, if $P(n)$ is true, then so is $P(n+1)$. This argument is valid for every nonnegative integer n, so this establishes the second fact required by the induction principle. Therefore, the induction principle says that the predicate $P(m)$ is true for all nonnegative integers, m, so the theorem is proved.

7.1.2 A Template for Induction Proofs

The proof of Theorem 7.1.1 was relatively simple, but even the most complicated induction proof follows exactly the same template. There are five components:

1. **State that the proof uses induction.** This immediately conveys the overall structure of the proof, which helps the reader understand your argument.
2. **Define an appropriate predicate $P(n)$.** The eventual conclusion of the induction argument will be that $P(n)$ is true for all nonnegative n. Thus, you should define the predicate $P(n)$ so that your theorem is equivalent to (or

follows from) this conclusion. Often the predicate can be lifted straight from the claim, as in the example above. The predicate $P(n)$ is called the *induction hypothesis*. Sometimes the induction hypothesis will involve several variables, in which case you should indicate which variable serves as n.

3. **Prove that $P(0)$ is true.** This is usually easy, as in the example above. This part of the proof is called the *base case* or *basis step*.

4. **Prove that $P(n)$ implies $P(n+1)$ for every nonnegative integer n.** This is called the *inductive step*. The basic plan is always the same: assume that $P(n)$ is true and then use this assumption to prove that $P(n+1)$ is true. These two statements should be fairly similar, but bridging the gap may require some ingenuity. Whatever argument you give must be valid for every nonnegative integer n, since the goal is to prove the implications $P(0) \to P(1)$, $P(1) \to P(2)$, $P(2) \to P(3)$, etc. all at once.

5. **Invoke induction.** Given these facts, the induction principle allows you to conclude that $P(n)$ is true for all nonnegative n. This is the logical capstone to the whole argument, but it is so standard that it's usual not to mention it explicitly,

Explicitly labeling the *base case* and *inductive step* may make your proofs clearer.

7.1.3 A Clean Writeup

The proof of Theorem 7.1.1 given above is perfectly valid; however, it contains a lot of extraneous explanation that you won't usually see in induction proofs. The writeup below is closer to what you might see in print and should be prepared to produce yourself.

Proof. We use induction. The induction hypothesis, $P(n)$, will be equation (7.1).

Base case: $P(0)$ is true, because both sides of equation (7.1) equal zero when $n = 0$.

Inductive step: Assume that $P(n)$ is true, where n is any nonnegative integer. Then

$$\begin{aligned} 1+2+3+\cdots+n+(n+1) &= \frac{n(n+1)}{2}+(n+1) && \text{(by induction hypothesis)} \\ &= \frac{(n+1)(n+2)}{2} && \text{(by simple algebra)} \end{aligned}$$

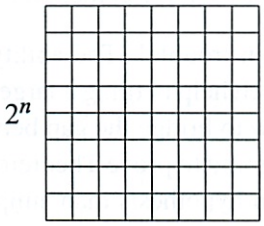
which proves $P(n+1)$.

So it follows by induction that $P(n)$ is true for all nonnegative n. ∎

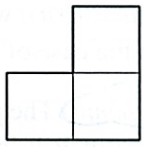
Induction was helpful for *proving the correctness* of this summation formula, but not helpful for *discovering* it in the first place. Tricks and methods for finding such formulas will appear in a later chapter.

7.1.4 Courtyard Tiling

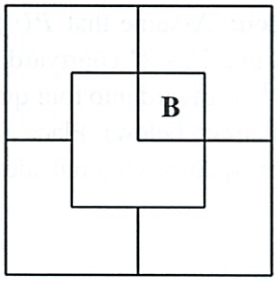
During the development of MIT's famous Stata Center, costs rose further and further over budget, and there were some radical fundraising ideas. One rumored plan was to install a big courtyard with dimensions $2^n \times 2^n$:



One of the central squares would be occupied by a statue of a wealthy potential donor. Let's call him "Bill". (In the special case $n = 0$, the whole courtyard consists of a single central square; otherwise, there are four central squares.) A complication was that the building's unconventional architect, Frank Gehry, was alleged to require that only special L-shaped tiles be used:



A courtyard meeting these constraints exists, at least for $n = 2$:



For larger values of n, is there a way to tile a $2^n \times 2^n$ courtyard with L-shaped tiles and a statue in the center? Let's try to prove that this is so.

Theorem 7.1.2. *For all $n \geq 0$ there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in a central square.*

Proof. (doomed attempt) The proof is by induction. Let $P(n)$ be the proposition that there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in the center.

Base case: $P(0)$ is true because Bill fills the whole courtyard.

Inductive step: Assume that there is a tiling of a $2^n \times 2^n$ courtyard with Bill in the center for some $n \geq 0$. We must prove that there is a way to tile a $2^{n+1} \times 2^{n+1}$ courtyard with Bill in the center .... ■

Now we're in trouble! The ability to tile a smaller courtyard with Bill in the center isn't much help in tiling a larger courtyard with Bill in the center. We haven't figured out how to bridge the gap between $P(n)$ and $P(n+1)$.

So if we're going to prove Theorem 7.1.2 by induction, we're going to need some *other* induction hypothesis than simply the statement about n that we're trying to prove.

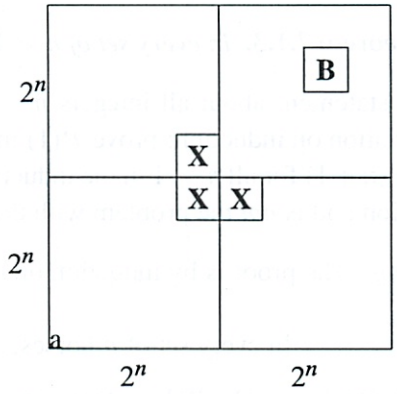
When this happens, your first fallback should be to look for a *stronger* induction hypothesis; that is, one which implies your previous hypothesis. For example, we could make $P(n)$ the proposition that for *every* location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder.

This advice may sound bizarre: "If you can't prove something, try to prove something grander!" But for induction arguments, this makes sense. In the inductive step, where you have to prove $P(n)$ IMPLIES $P(n+1)$, you're in better shape because you can *assume* $P(n)$, which is now a more powerful statement. Let's see how this plays out in the case of courtyard tiling.

Proof. (successful attempt) The proof is by induction. Let $P(n)$ be the proposition that for every location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder.

Base case: $P(0)$ is true because Bill fills the whole courtyard.

Inductive step: Assume that $P(n)$ is true for some $n \geq 0$; that is, for every location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder. Divide the $2^{n+1} \times 2^{n+1}$ courtyard into four quadrants, each $2^n \times 2^n$. One quadrant contains Bill (**B** in the diagram below). Place a temporary Bill (**X** in the diagram) in each of the three central squares lying outside this quadrant:



Now we can tile each of the four quadrants by the induction assumption. Replacing the three temporary Bills with a single L-shaped tile completes the job. This proves that $P(n)$ implies $P(n+1)$ for all $n \geq 0$. The theorem follows as a special case. ■

This proof has two nice properties. First, not only does the argument guarantee that a tiling exists, but also it gives an algorithm for finding such a tiling. Second, we have a stronger result: if Bill wanted a statue on the edge of the courtyard, away from the pigeons, we could accommodate him!

Strengthening the induction hypothesis is often a good move when an induction proof won't go through. But keep in mind that the stronger assertion must actually be *true*; otherwise, there isn't much hope of constructing a valid proof! Sometimes finding just the right induction hypothesis requires trial, error, and insight. For example, mathematicians spent almost twenty years trying to prove or disprove the conjecture that "Every planar graph is 5-choosable"[2]. Then, in 1994, Carsten Thomassen gave an induction proof simple enough to explain on a napkin. The key turned out to be finding an extremely clever induction hypothesis; with that in hand, completing the argument is easy!

7.1.5 A Faulty Induction Proof

False Theorem. *All horses are the same color.*

Notice that no n is mentioned in this assertion, so we're going to have to reformulate it in a way that makes an n explicit. In particular, we'll (falsely) prove that

[2]5-choosability is a slight generalization of 5-colorability. Although every planar graph is 4-colorable and therefore 5-colorable, not every planar graph is 4-choosable. If this all sounds like nonsense, don't panic. We'll discuss graphs, planarity, and coloring in a later chapter.

False Theorem 7.1.3. *In every set of $n \geq 1$ horses, all are the same color.*

This a statement about all integers $n \geq 1$ rather ≥ 0, so it's natural to use a slight variation on induction: prove $P(1)$ in the base case and then prove that $P(n)$ implies $P(n+1)$ for all $n \geq 1$ in the inductive step. This is a perfectly valid variant of induction and is *not* the problem with the proof below.

False proof. The proof is by induction on n. The induction hypothesis, $P(n)$, will be

$$\text{In every set of } n \text{ horses, all are the same color.} \tag{7.3}$$

Base case: ($n = 1$). $P(1)$ is true, because in a set of horses of size 1, there's only one horse, and this horse is definitely the same color as itself.

Inductive step: Assume that $P(n)$ is true for some $n \geq 1$. that is, assume that in every set of n horses, all are the same color. Now consider a set of $n+1$ horses:

$$h_1,\ h_2,\ \ldots,\ h_n,\ h_{n+1}$$

By our assumption, the first n horses are the same color:

$$\underbrace{h_1,\ h_2,\ \ldots,\ h_n}_{\text{same color}},\ h_{n+1}$$

Also by our assumption, the last n horses are the same color:

$$h_1,\ \underbrace{h_2,\ \ldots,\ h_n,\ h_{n+1}}_{\text{same color}}$$

So h_1 is the same color as the remaining horses besides h_{n+1}, and likewise h_{n+1} is the same color as the remaining horses besides h_1. So h_1 and h_{n+1} are the same color. That is, horses $h_1, h_2, \ldots, h_{n+1}$ must all be the same color, and so $P(n+1)$ is true. Thus, $P(n)$ implies $P(n+1)$.

By the principle of induction, $P(n)$ is true for all $n \geq 1$. ■

We've proved something false! Is math broken? Should we all become poets? No, this proof has a mistake.

The error in this argument is in the sentence that begins, "So h_1 and h_{n+1} are the same color." The "..." notation creates the impression that there are some remaining horses besides h_1 and h_{n+1}. However, this is not true when $n = 1$. In that case, the first set is just h_1 and the second is h_2, and there are no remaining horses besides them. So h_1 and h_2 need not be the same color!

This mistake knocks a critical link out of our induction argument. We proved $P(1)$ and we *correctly* proved $P(2) \longrightarrow P(3)$, $P(3) \longrightarrow P(4)$, etc. But we failed

to prove $P(1) \longrightarrow P(2)$, and so everything falls apart: we can not conclude that $P(2)$, $P(3)$, etc., are true. And, of course, these propositions are all false; there are horses of a different color.

Students sometimes claim that the mistake in the proof is because $P(n)$ is false for $n \geq 2$, and the proof assumes something false, namely, $P(n)$, in order to prove $P(n+1)$. You should think about how to explain to such a student why this claim would get no credit on a 6.042 exam.

7.2 Strong Induction

A useful variant of induction is called *strong induction*. Strong Induction and Ordinary Induction are used for exactly the same thing: proving that a predicate $P(n)$ is true for all $n \in \mathbb{N}$.

Principle of Strong Induction. Let $P(n)$ be a predicate. If

- $P(0)$ is true, and
- for all $n \in \mathbb{N}$, $P(0)$, $P(1)$, ..., $P(n)$ *together* imply $P(n+1)$,

then $P(n)$ is true for all $n \in \mathbb{N}$.

The only change from the ordinary induction principle is that strong induction allows you to assume more stuff in the inductive step of your proof! In an ordinary induction argument, you assume that $P(n)$ is true and try to prove that $P(n+1)$ is also true. In a strong induction argument, you may assume that $P(0)$, $P(1)$, ..., and $P(n)$ are *all* true when you go to prove $P(n+1)$. These extra assumptions can only make your job easier.

7.2.1 Products of Primes

As a first example, we'll use strong induction to re-prove Theorem 2.4.1 which we previously proved using Well Ordering.

Lemma 7.2.1. *Every integer greater than 1 is a product of primes.*

Proof. We will prove Lemma 7.2.1 by strong induction, letting the induction hypothesis, $P(n)$, be

$$n \text{ is a product of primes.}$$

So Lemma 7.2.1 will follow if we prove that $P(n)$ holds for all $n \geq 2$.

Base Case: ($n = 2$) $P(2)$ is true because 2 is prime, and so it is a length one product of primes by convention.

Inductive step: Suppose that $n \geq 2$ and that i is a product of primes for every integer i where $2 \leq i < n + 1$. We must show that $P(n + 1)$ holds, namely, that $n + 1$ is also a product of primes. We argue by cases:

If $n + 1$ is itself prime, then it is a length one product of primes by convention, so $P(n + 1)$ holds in this case.

Otherwise, $n + 1$ is not prime, which by definition means $n + 1 = km$ for some integers k, m such that $2 \leq k, m < n + 1$. Now by strong induction hypothesis, we know that k is a product of primes. Likewise, m is a product of primes. it follows immediately that $km = n$ is also a product of primes. Therefore, $P(n + 1)$ holds in this case as well.

So $P(n + 1)$ holds in any case, which completes the proof by strong induction that $P(n)$ holds for all nonnegative integers, n.

■

7.2.2 Making Change

The country Inductia, whose unit of currency is the Strong, has coins worth 3Sg (3 Strongs) and 5Sg. Although the Inductians have some trouble making small change like 4Sg or 7Sg, it turns out that they can collect coins to make change for any number that is at least 8 Strongs.

Strong induction makes this easy to prove for $n + 1 \geq 11$, because then $(n + 1) - 3 \geq 8$, so by strong induction the Inductians can make change for exactly $(n + 1) - 3$ Strongs, and then they can add a 3Sg coin to get $(n + 1)$Sg. So the only thing to do is check that they can make change for all the amounts from 8 to 10Sg, which is not too hard to do.

Here's a detailed writeup using the official format:

Proof. We prove by strong induction that the Inductians can make change for any amount of at least 8Sg. The induction hypothesis, $P(n)$ will be:

> If $n \geq 8$, then there is a collection of coins whose value is n Strongs.

Notice that $P(n)$ is an implication. When the hypothesis of an implication is false, we know the whole implication is true. In this situation, the implication is said to be *vacuously* true. So $P(n)$ will be vacuously true whenever $n < 8$.[3]

[3] Another approach that avoids these vacuous cases is to define

$$Q(n) ::= \text{there is a collection of coins whose value is } n + 8\text{Sg},$$

and prove that $Q(n)$ holds for all $n \geq 0$.

We now proceed with the induction proof:

Base case: $P(0)$ is vacuously true.

Inductive step: We assume $P(i)$ holds for all $i \leq n$, and prove that $P(n+1)$ holds. We argue by cases:

Case ($n+1 < 8$): $P(n+1)$ is vacuously true in this case.

Case ($n+1 = 8$): $P(8)$ holds because the Inductians can use one 3Sg coin and one 5Sg coins.

Case ($n+1 = 9$): Use three 3Sg coins.

Case ($n+1 = 10$): Use two 5Sg coins.

Case ($n+1 \geq 11$): Then $n \geq (n+1) - 3 \geq 8$, so by the strong induction hypothesis, the Inductians can make change for $(n+1)-3$ Strong. Now by adding a 3Sg coin, they can make change for $(n+1)$Sg.

So in any case, $P(n+1)$ is true, and we conclude by strong induction that for all $n \geq 8$, the Inductians can make change for n Strong.

■

7.2.3 The Stacking Game

Here is another exciting 6.042 game that's surely about to sweep the nation!

You begin with a stack of n boxes. Then you make a sequence of moves. In each move, you divide one stack of boxes into two nonempty stacks. The game ends when you have n stacks, each containing a single box. You earn points for each move; in particular, if you divide one stack of height $a+b$ into two stacks with heights a and b, then you score ab points for that move. Your overall score is the sum of the points that you earn for each move. What strategy should you use to maximize your total score?

As an example, suppose that we begin with a stack of $n = 10$ boxes. Then the game might proceed as follows:

Stack Heights										**Score**
$\underline{10}$										
5	$\underline{5}$									25 points
$\underline{5}$	3	2								6
$\underline{4}$	3	2	1							4
2	$\underline{3}$	2	1	2						4
$\underline{2}$	2	2	1	2	1					2
1	$\underline{2}$	2	1	2	1	1				1
1	1	$\underline{2}$	1	2	1	1	1			1
1	1	1	1	$\underline{2}$	1	1	1	1		1
1	1	1	1	1	1	1	1	1	1	1
								Total Score	=	45 points

On each line, the underlined stack is divided in the next step. Can you find a better strategy?

Analyzing the Game

Let's use strong induction to analyze the unstacking game. We'll prove that your score is determined entirely by the number of boxes —your strategy is irrelevant!

Theorem 7.2.2. *Every way of unstacking n blocks gives a score of $n(n-1)/2$ points.*

There are a couple technical points to notice in the proof:

- The template for a strong induction proof is exactly the same as for ordinary induction.
- As with ordinary induction, we have some freedom to adjust indices. In this case, we prove $P(1)$ in the base case and prove that $P(1), \ldots, P(n)$ imply $P(n+1)$ for all $n \geq 1$ in the inductive step.

Proof. The proof is by strong induction. Let $P(n)$ be the proposition that every way of unstacking n blocks gives a score of $n(n-1)/2$.

Base case: If $n = 1$, then there is only one block. No moves are possible, and so the total score for the game is $1(1-1)/2 = 0$. Therefore, $P(1)$ is true.

Inductive step: Now we must show that $P(1), \ldots, P(n)$ imply $P(n+1)$ for all $n \geq 1$. So assume that $P(1), \ldots, P(n)$ are all true and that we have a stack of $n+1$ blocks. The first move must split this stack into substacks with positive sizes a and b where $a + b = n + 1$ and $0 < a, b \leq n$. Now the total score for the game is the sum of points for this first move plus points obtained by unstacking the two resulting substacks:

$$\begin{aligned}
\text{total score} &= (\text{score for 1st move}) \\
&\quad + (\text{score for unstacking } a \text{ blocks}) \\
&\quad + (\text{score for unstacking } b \text{ blocks}) \\
&= ab + \frac{a(a-1)}{2} + \frac{b(b-1)}{2} && \text{by } P(a) \text{ and } P(b) \\
&= \frac{(a+b)^2 - (a+b)}{2} = \frac{(a+b)((a+b)-1)}{2} \\
&= \frac{(n+1)n}{2}
\end{aligned}$$

This shows that $P(1), P(2), \ldots, P(n)$ imply $P(n+1)$.

Therefore, the claim is true by strong induction. ∎

Despite the name, strong induction is technically no more powerful than ordinary induction, though it makes some proofs easier to follow. But any theorem that can be proved with strong induction could also be proved with ordinary induction (using a slightly more complicated induction hypothesis). On the other hand, announcing that a proof uses ordinary rather than strong induction highlights the fact that $P(n+1)$ follows directly from $P(n)$, which is generally good to know.

7.3 Induction versus Well Ordering

The Induction Axiom looks nothing like the Well Ordering Principle, but these two proof methods are closely related. In fact, as the examples above suggest, we can take any Well Ordering proof and reformat it into an Induction proof. Conversely, it's equally easy to take any Induction proof and reformat it into a Well Ordering proof.

So what's the difference? Well, sometimes induction proofs are clearer because they resemble recursive procedures that reduce handling an input of size $n+1$ to handling one of size n. On the other hand, Well Ordering proofs sometimes seem more natural, and also come out slightly shorter. The choice of method is really a matter of style—but style does matter.

Problems for Section 7.1

Class Problems

Problem 7.1.
Use induction to prove that

$$1^3 + 2^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2. \tag{7.4}$$

for all $n \geq 1$.

Remember to formally

1. Declare proof by induction.
2. Identify the induction hypothesis $P(n)$.
3. Establish the base case.
4. Prove that $P(n) \Rightarrow P(n+1)$.
5. Conclude that $P(n)$ holds for all $n \geq 1$.

as in the five part template.

Problem 7.2.
Prove by induction on n that

$$1 + r + r^2 + \cdots + r^n = \frac{r^{n+1} - 1}{r - 1} \tag{7.5}$$

for all $n \in \mathbb{N}$ and numbers $r \neq 1$.

Problem 7.3.
Prove by induction:

$$1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} < 2 - \frac{1}{n}, \tag{7.6}$$

for all $n > 1$.

Problem 7.4. **(a)** Prove by induction that a $2^n \times 2^n$ courtyard with a 1×1 statue of Bill in *a corner* can be covered with L-shaped tiles. (Do not assume or reprove the (stronger) result of Theorem 7.1.2 that Bill can be placed anywhere. The point of this problem is to show a different induction hypothesis that works.)

(b) Use the result of part (a) to prove the original claim that there is a tiling with Bill in the middle.

Problem 7.5.
Find the flaw in the following bogus proof that $a^n = 1$ for all nonnegative integers n, whenever a is a nonzero real number.

Bogus proof. The proof is by induction on n, with hypothesis

$$P(n) ::= \forall k \leq n. a^k = 1,$$

where k is a nonnegative integer valued variable.

Base Case: $P(0)$ is equivalent to $a^0 = 1$, which is true by definition of a^0. (By convention, this holds even if $a = 0$.)

Inductive Step: By induction hypothesis, $a^k = 1$ for all $k \in \mathbb{N}$ such that $k \leq n$. But then

$$a^{n+1} = \frac{a^n \cdot a^n}{a^{n-1}} = \frac{1 \cdot 1}{1} = 1,$$

which implies that $P(n+1)$ holds. It follows by induction that $P(n)$ holds for all $n \in \mathbb{N}$, and in particular, $a^n = 1$ holds for all $n \in \mathbb{N}$.

■

Problem 7.6.
We've proved in two different ways that

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

But now we're going to prove a *contradictory* theorem!

False Theorem. *For all $n \geq 0$,*

$$2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2}$$

Proof. We use induction. Let $P(n)$ be the proposition that $2 + 3 + 4 + \cdots + n = n(n+1)/2$.
Base case: $P(0)$ is true, since both sides of the equation are equal to zero. (Recall that a sum with no terms is zero.)
Inductive step: Now we must show that $P(n)$ implies $P(n+1)$ for all $n \geq 0$. So suppose that $P(n)$ is true; that is, $2 + 3 + 4 + \cdots + n = n(n+1)/2$. Then we can reason as follows:

$$\begin{aligned} 2 + 3 + 4 + \cdots + n + (n+1) &= [2 + 3 + 4 + \cdots + n] + (n+1) \\ &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{(n+1)(n+2)}{2} \end{aligned}$$

Above, we group some terms, use the assumption $P(n)$, and then simplify. This shows that $P(n)$ implies $P(n+1)$. By the principle of induction, $P(n)$ is true for all $n \in \mathbb{N}$. ■

Where exactly is the error in this proof?

Homework Problems

Problem 7.7.

Claim 7.3.1. *If a collection of positive integers (not necessarily distinct) has sum $n \geq 1$, then the collection has product at most $3^{n/3}$.*

For example, the collection 2, 2, 3, 4, 4, 7 has the sum:

$$2+2+3+4+4+7=22$$

On the other hand, the product is:

$$\begin{aligned} 2\cdot 2\cdot 3\cdot 4\cdot 4\cdot 7 &= 1344 \\ &\leq 3^{22/3} \\ &\approx 3154.2 \end{aligned}$$

(a) Use strong induction to prove that $n \leq 3^{n/3}$ for every integer $n \geq 0$.

(b) Prove the claim using induction or strong induction. (You may find it easier to use induction on the *number of positive integers in the collection* rather than induction on the sum n.)

Problem 7.8.

For any binary string, α, let $\text{num}(\alpha)$ be the nonnegative integer it represents in binary notation. For example, $\text{num}(10) = 2$, and $\text{num}(0101) = 5$.

An $n+1$-*bit adder* adds two $n+1$-bit binary numbers. More precisely, an $n+1$-bit adder takes two length $n+1$ binary strings

$$\begin{aligned} \alpha_n &::= a_n \ldots a_1 a_0, \\ \beta_n &::= b_n \ldots b_1 b_0, \end{aligned}$$

and a binary digit, c_0, as inputs, and produces a length $n+1$ binary string

$$\sigma_n ::= s_n \ldots s_1 s_0,$$

and a binary digit, c_{n+1}, as outputs, and satisfies the specification:

$$\text{num}(\alpha_n) + \text{num}(\beta_n) + c_0 = 2^{n+1} c_{n+1} + \text{num}(\sigma_n). \tag{7.7}$$

There is a straighforward way to implement an $n+1$-bit adder as a digital circuit: an $n+1$-bit *ripple-carry circuit* has $1+2(n+1)$ binary inputs

$$a_n, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0, c_0,$$

and $n+2$ binary outputs,

$$c_{n+1}, s_n, \ldots, s_1, s_0.$$

As in Problem 3.4, the ripple-carry circuit is specified by the following formulas:

$$s_i ::= a_i \text{ XOR } b_i \text{ XOR } c_i \tag{7.8}$$

$$c_{i+1} ::= (a_i \text{ AND } b_i) \text{ OR } (a_i \text{ AND } c_i) \text{ OR } (b_i \text{ AND } c_i),. \tag{7.9}$$

for $0 \leq i \leq n$.

(a) Verify that definitions (7.8) and (7.9) imply that

$$a_n + b_n + c_n = 2c_{n+1} + s_n. \tag{7.10}$$

for all $n \in \mathbb{N}$.

(b) Prove by induction on n that an $n+1$-bit ripple-carry circuit really is an $n+1$-bit adder, that is, its outputs satisfy (7.7).

Hint: You may assume that, by definition of binary representation of integers,

$$\text{num}(\alpha_{n+1}) = a_{n+1}2^{n+1} + \text{num}(\alpha_n). \tag{7.11}$$

Problem 7.9.

The Math for Computer Science mascot, Theory Hippotamus, made a startling discovery while playing with his prized collection of unit squares over the weekend. Here is what happened.

First, Theory Hippotamus put his favorite unit square down on the floor as in Figure 7.1 (a). He noted that the length of the periphery of the resulting shape was 4, an even number. Next, he put a second unit square down next to the first so that the two squares shared an edge as in Figure 7.1 (b). He noticed that the length of the periphery of the resulting shape was now 6, which is also an even number. (The periphery of each shape in the figure is indicated by a thicker line.) Theory Hippotamus continued to place squares so that each new square shared an edge with at least one previously-placed square and no squares overlapped. Eventually, he arrived at the shape in Figure 7.1 (c). He realized that the length of the periphery of this shape was 36, which is again an even number.

Our plucky porcine pal is perplexed by this peculiar pattern. Use induction on the number of squares to prove that the length of the periphery is always even, no matter how many squares Theory Hippotamus places or how he arranges them.

f98ps2-c.pdf

Redrawn

f98ps2-b.pdf

Redrawn

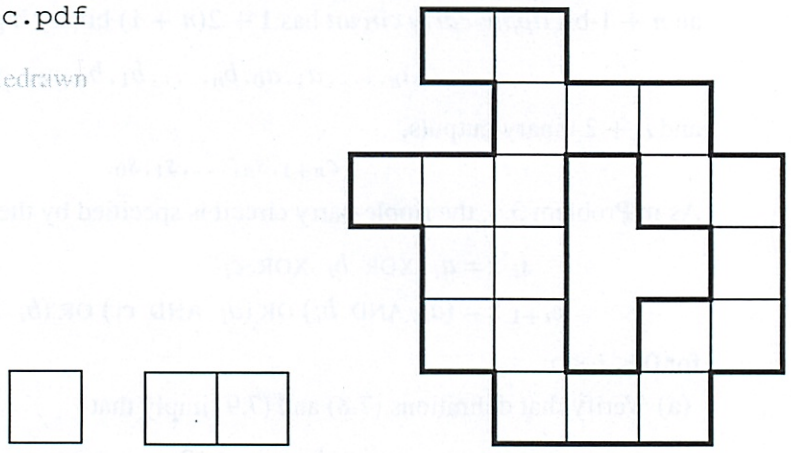


Figure 7.1 Some shapes that Theory Hippotamus created.

Problems for Section 7.2

Class Problems

Problem 7.10.
A group of $n \geq 1$ people can be divided into teams, each containing either 4 or 7 people. What are all the possible values of n? Use induction to prove that your answer is correct.

Problem 7.11.
The following Lemma is true, but the *proof* given for it below is defective. Pinpoint *exactly* where the proof first makes an unjustified step and explain why it is unjustified.

Lemma 7.3.2. *For any prime p and positive integers $n, x_1, x_2, \ldots, x_n$, if $p \mid x_1 x_2 \ldots x_n$, then $p \mid x_i$ for some $1 \leq i \leq n$.*

Bogus proof. Proof by strong induction on n. The induction hypothesis, $P(n)$, is that Lemma holds for n.

Base case $n = 1$: When $n = 1$, we have $p \mid x_1$, therefore we can let $i = 1$ and conclude $p \mid x_i$.

Induction step: Now assuming the claim holds for all $k \leq n$, we must prove it for $n + 1$.

So suppose $p \mid x_1 x_2 \ldots x_{n+1}$. Let $y_n = x_n x_{n+1}$, so $x_1 x_2 \ldots x_{n+1} = x_1 x_2 \ldots x_{n-1} y_n$. Since the righthand side of this equality is a product of n terms, we have by induction that p divides one of them. If $p \mid x_i$ for some $i < n$, then we have the desired i. Otherwise $p \mid y_n$. But since y_n is a product of the two terms x_n, x_{n+1}, we have by strong induction that p divides one of them. So in this case $p \mid x_i$ for $i = n$ or $i = n + 1$. ■

Problem 7.12.

Define the *potential*, $p(S)$, of a stack of blocks, S, to be $k(k-1)/2$ where k is the number of blocks in S. Define the potential, $p(A)$, of a set of stacks, A, to be the sum of the potentials of the stacks in A.

Generalize Theorem 7.2.2 about scores in the stacking game to show that for any set of stacks, A, if a sequence of moves starting with A leads to another set of stacks, B, then $p(A) \geq p(B)$, and the score for this sequence of moves is $p(A) - p(B)$.

Hint: Try induction on the number of moves to get from A to B.