Lecture Notes for CSCI C241/H241

Induction, Recursion and Programming[*]

Steven D. Johnson
Computer Science Department
Indiana University School of Informatics

[*]Induction, Recursion, and Programming *is a Working title. Some content derives from the book* Induction, Recursion, and Programming *by Mitchell Wand (North-Holland, 1976). All rights to original content are reserved by Mitchell Wand.*

Contents

Chapter 1

Sets

The concepts of *set* and *set membership* are fundamental. A set is determined by its *elements* (or *members*) and to say that x is an element of the set S, we write:

$$x \in S$$

To say that y is *not* a member of S we write:

$$y \notin S$$

Simple sets are specified by listing all of their elements between braces. A set A of five numbers is specified:

$$A = \{1, 2, 3, 4, 5\}$$

A set B of three colors is specified:

$$B = \{\mathit{red},\ \mathit{blue},\ \mathit{green}\}$$

A set C of three symbolic color names is specified

$$C = \{\texttt{red},\ \texttt{blue},\ \texttt{green}\}$$

This book uses `teletype` font for *concrete syntax*, the purely typographical form of a symbol.

Example 1.1 List the set S of whole numbers between 1 and 15 which are evenly divisible by either 2 or 3.

Solution: We might begin by listing the numbers divisible by 2:

$$\{2, 4, 6, 8, 10, 12, 14, \ldots$$

and then the numbers divisible by 3:

$$\ldots, 3, 6, 9, 12, 15\}$$

So the set we are looking for could be written

$$S = \{2, 4, 6, 8, 10, 12, 14, 3, 6, 9, 12, 15\}$$

This listing contains duplications and the numbers are listed in a strange order. Neither of these problems makes the description incorrect, but it is less confusing to list each element just once. A better description for S is

$$\{2, 3, 4, 6, 8, 9, 10, 12, 14, 15\}$$

$\square$

One way to abbreviate a long list of elements is to use *ellipses* to indicate a large, possibly infinite, range of values. For example, the set of lower-case letters from `a` to `z` could be expressed as follows:

$$\{\texttt{a},\ \texttt{b},\ \ldots,\ \texttt{z}\}$$

The set of numbers ranging from 1 to 10,000 could be specified:

$$\{1, 2, 3, \ldots, 10000\}$$

The following definition uses ellipses to describe some infinite sets that are used throughout this book.

Definition 1.1

(a) *The set of* whole numbers *is* $\mathbb{W} = \{1, 2, 3, \ldots\}$.

(b) *The set of* natural numbers *is* $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$

(c) *The set of* integers *is* $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$

Ellipses are useful when it is natural to list a set's elements in some consecutive order, but care is needed in their use. Consider the set specification:

$$B = \{2, 4, \ldots, 64\}$$

It is not *completely* obvious from this set definition whether the elements of B are:

- the even numbers from 2 to 64:

$$\begin{aligned} B \overset{?}{=} \{\ & 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, \\ & 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64\ \}, \end{aligned}$$

- or the powers-of-2 from 2 to 64:

$$B \overset{?}{=} \{2, 4, 8, 16, 32, 64\},$$

- or something else.

The rule-of-thumb is to take the simplest sequence that clearly describes the set exactly, but "simplest" is a matter of judgment involving assumptions about the knowledge of the Reader. One way to avoid confusion is to include a formula representing a typical element of the list; for example:

$$B = \{2, \ldots, 2i, \ldots, 64\}$$

This description says that the elements of B have the form $2i$; they are even numbers. Although even this definition relies on the reader to understand that i refers to a whole number, the specification is better determined.

1.1 Set Builder Notation

A more general way to specify the elements of a set is to write down a property that is satisfied by the elements of the set—and only by those elements. The notation for doing this is called *set builder notation*:

$$\{x \in U \mid P[x]\}$$

U is the *universe* or *domain* from which prospective elements x originate, and $P[x]$ stands for some property that the members—and only the members—must satisfy. The property P must "make sense" with respect to U; that is, is, $P[u]$ must be either *true* or *false* for every element of U. The set specified contains *all* of the elements for which $P[x]$ is *true.* Mention of U may be omitted if either the surrounding context or P make clear what U is.

For example, the second version of B above might be specified

$$B = \{x \in \mathbb{W} \mid x = 2^i \text{ for } i \in \mathbb{W} \text{ such that } 1 \leq i \leq 6\}$$

The property $P[x]$ in this case is

$$P[x] \equiv \text{“}x = 2^i \text{ for some } i \in \mathbb{W} \text{ such that } 1 \leq i \leq 6\text{”}$$

Even though P contains two variables, x and i, it is nevertheless a statement about x *only; P quantifies* i, stating $i \in \{1, 2, 3, 4, 5, 6\}$.

In this instance, the set-builder description is not much of an an improvement over simply listing the elements, although if i ranged from 1 to 100 it would be. We can do a little better by writing a formula in place of the simple variable x:

$$B = \{2^i \mid i \in \mathbb{W} \text{ and } 1 \leq i \leq 6\}$$

we have omitted the declaration "$2^i \in \mathbb{W}$" because it is clear from the context of the example that B is a set of whole numbers.

Remark. You may have noticed that property P, above, is defined using '[', ']' and '$\equiv$', rather than '(', ')' and '='. There is no difference in meaning, but throughout this book the $P[x] \equiv$ " $\cdots, \cdot$ " notation is used for the purpose of linguistic identification, that is, naming formulas rather than the values they denote.

Example 1.2 Specify the infinite set E of *nonnegative even numbers* using ellipses and then again using set-builder notation

SOLUTION: Using ellipses, one could write $E = \{0, 2, 4, 6, \ldots\}$ It would be clearer to include a representative element $E = \{0, 2, \ldots,\ 2i,\ \ldots\}$. Using set-builder notation, we would write $\{2n \mid n \in \mathbb{N}\}$. □

Here are some other numerical set names used in this book.

Definition 1.2

(a) *The set of* integers modulo n *is* $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$.

(b) *The set of* rational numbers, $\mathbb{Q}$, *consists of all fractions:*

$$\mathbb{Q} = \{\frac{n}{m} \mid n \in \mathbb{N} \textit{ and } m \in \mathbb{W}\}$$

(c) *The* real numbers,[†] $\mathbb{R}$.

1.2 Set Operations

There is that set which contains no elements. One way to express this set is to place nothing between braces: $\{\}$. We also use the symbol $\emptyset$ for this set.

Definition 1.3 *The* empty set, *denoted by $\emptyset$, has no elements: $\emptyset = \{\}$.*

A common mistake is writing "$\{\emptyset\}$" for the empty set. However, $\{\emptyset\}$ is *not* the "really empty" set but rather a set with a single element, namely, $\emptyset$.

Sets are compared by asking what elements they have in common.

Definition 1.4 *Let A and B be two sets.*

(a) *A* equals *B, written $A = B$, if A and B contain exactly the same elements.*

(b) *A* contains *B, written $B \subseteq A$, if every element of B is also an element of A. A is said to* properly contain *B when $B \subseteq A$ and $B \neq A$. This is sometimes written as $B \subsetneq A$.*

(c) *A and B are* disjoint *when they have no elements in common, that is, $A \cap B = \emptyset$.*

[†]Despite the fact we have learned about and worked with real numbers most of our lives, it is hard to state a concise property explaining just what a real number is, $\mathbb{R} = \{x \mid R[x]\}$. Two dictionary definitions are

1. Every real number can be expressed with an infinite decimal expansion, for instance, $\frac{1}{3} = 0.333\ldots$.
2. Every real number is the limit of an infinte sequence of rational numbers.

While true, both of these properties raise more questions than they answer.

To prove that two sets, A and B are equal, one often shows that each contains the other. The following proposition follows immediately from Definition 1.4.

Fact 1.1 *$A = B$ iff $A \subseteq B$ and $B \subseteq A$.*

There are numerous ways to build new sets from sets which are given. The most common of these have names and special symbols associated with them as defined below.

Definition 1.5 *Let A and B be two sets.*

(a) *The* intersection *of A and B, written $A \cap B$, is the collection of elements that A and B have in common. That is,*

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

(b) *The* union *of A and B, written $A \cup B$, is the collection of all those elements in either set or both. That is,*

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

(c) *The* (set) difference *of A and B, written $A \setminus B$ is the collection of those elements of A which are not in B. That is,*

$$A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$$

(d) *The* power set *of A written $\mathcal{P}(A)$, is the collection of A's subsets. That is,*

$$\mathcal{P}(A) = \{S \mid S \subseteq A\}$$

(e) *The* product *of A and B, written $A \times B$ is the collection of all* ordered pairs *whose first elements come from A and whose second elements come from B. That is,*

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

Example 1.3 Let A be the set $\{1, 5\}$; and let B be the set $\{1, 2, 3\}$. List the sets $A \cap B$, $A \cup B$, $A \setminus B$, $\mathcal{P}(A)$, $A \times B$, $B \times A$, and $A \times \mathbb{N}$.

Solution: The sets are

$$\begin{aligned}
A \cap B &= \{1\} \\
A \cup B &= \{1, 2, 3, 5\} \\
A \setminus B &= \{5\} \\
\mathcal{P}(A) &= \{\emptyset, \{1\}, \{5\}, \{1, 5\}\} \\
A \times B &= \{(1, 1), (1, 2), (1, 3), (5, 1), (5, 2), (5, 3)\} \\
B \times A &= \{(1, 1), (1, 5), (2, 1), (2, 5), (3, 1), (3, 5)\} \\
A \times \mathbb{N} &= \{(1, n) \mid n \in \mathbb{N}\} \cup \{(5, n) \mid n \in \mathbb{N}\} \\
&= \{(1, 0), (5, 0), (1, 1), (5, 1), \ldots (1, i), (5, i), \ldots\}
\end{aligned}$$

In listing the elements of $A \cup B$, each distinct element is written just once. Consult Definition 1.4 to verify that each element of $\mathcal{P}(A)$ is, in fact, a subset of A.

Ordered pairs $(1,3)$ and $(3,1)$ are unequal, for while they contain the same numbers, these numbers are in a different order. Thus, $A \times B$ and $B \times A$ are distinct sets because, for instance, $(1,3) \in A \times B$ but $(1,3) \notin B \times A$. However $A \times B$ and $B \times A$ are not disjoint; they share the element $(1,1)$. □

Example 1.4 Let $A = \{a, b\}$; let $B = \{0, 1\}$; and let $C = \{1, 3\}$. Compare the sets $(A \times B) \times C$ and $A \times (B \times C)$

Solution: First,

$$A \times B = \{(a,0), (a,1), (b,0), (b,1)\}$$

Now, the set $(A \times B) \times C$ is a set of ordered pairs, each of which has an ordered pair in its first position:

$$\begin{aligned}(A \times B) \times C = \{&((a,0),1), ((a,1),1), ((b,0),1), ((b,1),1),\\ &((a,0),3), ((a,1),3), ((b,0),3), ((b,1),3)\ \}\end{aligned}$$

Elements of the set $A \times (B \times C)$ have the same "information content" but a different structure:

$$\begin{aligned}A \times (B \times C) = \{&(a,(0,1)), (a,(1,1)), (b,(0,1)), (b,(1,1)),\\ &(a,(0,3)), (a,(1,3)), (b,(0,3)), (b,(1,3))\ \}\end{aligned}$$

Each of the ordered pairs in $A \times (B \times C)$ has its first element coming from A and its second element coming from $B \times C$. □

As Example 1.4 illustrates, compound set products may introduce structure that is not wanted. The next definition extends the notion of "product" to an arbitrary number of sets, as well as other variations of the "set-$\times$" operation.

Definition 1.6 *The* product of n sets*,* A_1*,* A_2*, …,* A_n*, is*

$$A_1 \times A_2 \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i,\ 1 \le i \le n\}$$

The elements of $A_1 \times A_2 \times A_3$ *are called* ordered n-tuples*. The* n*-fold product* A^n *is*

$$A^n = \overbrace{A \times \cdots \times A}^{n \text{ times}}$$

By convention, A^0 *is the empty set.*

Exercises 1.2

1. List the following sets:

(a) $\{2^i \mid i \in \mathbb{N} \text{ and } 0 \leq i \leq 8\}$

(b) $\{i^2 \mid i \in \mathbb{N} \text{ and } 0 \leq i \leq 8\}$

(c) $\{2k+1 \mid k \in \mathbb{N}\}$

(d) $\{m \mid 23 < m < 29 \text{ and } m \text{ is a prime number}\}$

2. Let $A = \{a, b\}$; let $B = \{1, 2, 3\}$; let $C = \emptyset$; and let $D = \{a, b, c, d\}$. List the following sets:

(a)	$A \cup B$	(f)	$A \cup C$
(b)	$A \cap B$	(g)	$A \cap D$
(c)	$A \times B$	(h)	A^3
(d)	$\mathcal{P}(A)$	(i)	$\mathcal{P}(\emptyset)$
(e)	$B \times \emptyset$	(j)	$(D \cap A) \times B$

3. Let $A = \{a, b\}$; let $B = \{1, 2, 3\}$; and let $E = A \times B$. List the following sets:

(a) $\{(x, y, y) \mid (x, y) \in E\}$

(b) $\{(x, x) \mid x \in E\}$

(c) $\{(y, z) \mid (x, y) \in E \text{ and } z \in B\}$

4. In this book, the set $S = \{1, 2, 2, 3, 3\}$ denotes a three-element set in which 2 and 3 have both been listed twice. So we understand that this another way to describe the set $\{1, 2, 3\}$. Some other books interpret $\{1, 2, 2, 3, 3\}$ as a *five* element *mulitset* allowing multiple occurances of equal elements. Write a version of Definition 1.5 for multisets.

5. Define a set P_n representing the prime divisors of a number n.

Comment: A simple set of numbers does not work because the convention is to allow listing redundant elements. So if one were to define P_{72} to be $\{2, 2, 2, 3, 3\}$, the specified set is actually just $\{2, 3\}$.

The question is asking you to devise a way, using just sets and set operations, to "represent" a number's prime decomposition. Unless you know more about how you are going to *use* this representation, there is no best way to do it. Nevertheless, you should take "elegance," (economy of expression, utility of notation) into consideration.

1.3 Languages

Any set can be thought of as an *alphabet* of *symbols* from which we can build the words, phrases, and sentences of a *language*.

Definition 1.7 *Let V be any set. The set of* words over V*, denoted by V^+, consists of all finite sequences of symbols from V. V is called the* alphabet *for V^+.*

Example 1.5 Let $V = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ and list V^+.
Solution: The set of words over V is

$$\begin{aligned} V^+ = \{ \; & \texttt{a}, \texttt{b}, \texttt{c}, \\ & \texttt{aa}, \texttt{ab}, \texttt{ac}, \texttt{ba}, \texttt{bb}, \texttt{bc}, \texttt{ca}, \texttt{cb}, \texttt{cc}, \\ & \texttt{aaa}, \texttt{aab}, \texttt{aac}, \texttt{aba}, \texttt{abb}, \texttt{abc}, \texttt{aca}, \texttt{acb}, \texttt{acc}, \\ & \texttt{baa}, \texttt{bab}, \texttt{bac}, \texttt{bba}, \texttt{bbb}, \texttt{bbc}, \texttt{bca}, \texttt{bcb}, \texttt{bcc}, \\ & \texttt{caa}, \texttt{cab}, \texttt{cac}, \texttt{cba}, \texttt{cbb}, \texttt{cbc}, \texttt{cca}, \texttt{ccb}, \texttt{ccc}, \\ & \texttt{aaaa}, \texttt{aaab}, \texttt{aaac}, \texttt{aaba}, \ldots, \texttt{cccc}, \\ & \vdots \end{aligned}$$

This listing shows all the one-letter words in the first line, all the two-letter words in the second line, and so forth. Within each line, the words listed systematically, in "alphabetic" order. □

Example 1.6 Let $W = \{5, 17\}$. List W^+. Solution: In the description below, an explicit *concatenation* mark is used to set individual symbols apart.

$$\begin{aligned} W^+ = \{ & 5,\ 17,\ 5\hat{}5,\ 5\hat{}17,\ 17\hat{}5,\ 17\hat{}17,\ 5\hat{}5\hat{}5,\ 5\hat{}5\hat{}17, 5\hat{}17\hat{}5,\ 5\hat{}17\hat{}17, \\ & \ldots\} \end{aligned}$$

For instance the word $17\hat{}5\hat{}5$ is composed of the three symbols 17, 5, and 5 from W.

The concatenation symbol is omitted—unless to do so causes confusion—just as the multiplication symbol is omitted in arithmetic formulas. It should be clear that the concatenation of two words is a word. □

Definition 1.8 *The* concatenation *of words u and v, is the word formed by juxtaposing u and v, that is, first spelling u and then spelling v; This new word is denoted by $u\hat{}v$, or where possible, simply by uv. We write u^n for*

$$\overbrace{u\hat{}u\hat{}\cdots\hat{}u}^{n \text{ times}}$$

Example 1.7 Let $V = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ and consider V^+, as defined in Example 1.5. Let $u = \texttt{aab}$, $v = \texttt{cc}$, and $w = \texttt{abc}$.

$$\begin{aligned}
\texttt{cca}\hat{\ }\texttt{abb} &= \texttt{ccaabb} \\
\texttt{a}\hat{\ }\texttt{bbbb}\hat{\ }\texttt{a} &= \texttt{abbbba} \\
(\texttt{a}\hat{\ }\texttt{c})u &= \texttt{acaab} \\
v\,\texttt{b}\,v &= \texttt{ccbcc} \\
uvw &= \texttt{aabccabc} \\
uuu = u^3 &= \texttt{aabaabaab}
\end{aligned}$$

□

If u, v, and w are words, then

$$u\hat{\ }(v\hat{\ }w) = (u\hat{\ }v)\hat{\ }w$$

In other words, when three or more words are concatenated, it does not matter whether the concatenation is done from left to right or from right to left (or in any other way) so long as the order is preserved.

It is sometimes useful to include a "word" containing no letters. The following definition provides a symbol for this word.

Definition 1.9 *The* empty word*, over any alphabet, is denoted by ε. Given alphabet V, for any word $w \in V^+$,*

$$\varepsilon\hat{\ }w = w \quad \text{and} \quad w\hat{\ }\varepsilon = w$$

The language V^* includes all words in V^+ together with ε,

$$V^* = V^+ \cup \{\varepsilon\}$$

Usually, we are interested in some particular subset of words over an alphabet.

Definition 1.10 *A* language *over alphabet V is any subset of V^*.*

Example 1.8 Describe the language of *decimal numerals* as might be accepted in a programming language.

SOLUTION: Let

$$D = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}\}$$

represent the set of digits. We will also need punctuation symbols from the set

$$P = \{\texttt{.},\ \texttt{+},\ \texttt{-}\}$$

The components of a numeral are

(a) A *sign*, which may be omitted for a positive number. Hence the sign comes from the set

$$S = \{\texttt{+},\ \texttt{-},\ \varepsilon\}$$

(b) the integer part, a string of one or more digits, D^+.

(c) the fractional part, if present, is a period followed by a string of zero or more digits,

$$E = \{\varepsilon\} \cup \{\,\texttt{.}\,\hat{}\,f \mid f \in D^*\}$$

Putting these together, the language of decimal numerals is described by

$$\mathit{Numerals} = \{s\,\hat{}\,m\,\hat{}\,f \mid s \in S,\ m \in D^+, \text{ and } f \in E\}$$

Here are some word instances in *Numeral*:

(a) The word `+2.731` is a valid numeral. It breaks down into a sign, integral part, and factional part according to the specification expression

$$\underbrace{\boxed{\texttt{+}}}_{S}\,\hat{}\,\underbrace{\boxed{\texttt{2}}}_{D^*}\,\hat{}\,\underbrace{\boxed{\texttt{.735}}}_{E}$$

(b) The word `42` is also valid, having an empty sign and fractional part

$$\boxed{\varepsilon}\boxed{\texttt{42}}\boxed{\varepsilon}$$

(c) The words `55.126.99`, `++5` and ε do not satisfy the description and hence are not in the language intended.

□

Exercises 1.3

1. Let $V = \{\texttt{a}, \texttt{b}, \texttt{\$}\}$. For each of the following languages $L_i \subseteq V^+$, list enough elements to make it clear what each contains.

(a) In language L_1 each word has exactly one `$` and equally many `a` s as `b` s.

(b) In each word of language L_2, `a` s and `b` s alternate with any number of `$` s mixed in.

(c) In each word of language L_3, no `a` occurs next to a `b` .

(d) $L_4 = \{u\,\hat{}\,\texttt{\$}\,\hat{}\,v \mid u \in \{\texttt{a}\}^+ \text{ and } v \in \{\texttt{\$}, \texttt{b}\}^+\}$

(e) $L_5 = \{\texttt{a}^k\,\hat{}\,\texttt{\$}\,\hat{}\,\texttt{b}^k \mid k \in \mathbb{N}\}$

2. In Exercise 1.5 the listing of $\{\texttt{a}, \texttt{b}, \texttt{c}\}^+$ shows that there are 3 one-letter words and 9 two-letter words. The third and fourth rows of the listing do not show all the possible words. How many words would there be in the third row; that is, how many three-letter words are there in $\{\texttt{a}, \texttt{b}, \texttt{c}\}^+$? How many four-letter words? How many n-letter words?

1.4 A Simple Algorithmic Language

A very simple programming language is used later this book. We introduce the language informally in this section; in Chapter **??**, once we have the needed mathematical foundations, we examine this language in more detail. It is a structured, sequential *language of statements*, similar in form to many languages that exist today like C and Java. It is assumed that you have some experience with, and intuition about, programming in this kind of language.

There are just four kinds of statements.

1. The *assignment statement*, has the form

   $$v := E$$

   The object to the left of the assignment symbol is called an *identifier*, or sometimes *program variable* (but never just *"variable"*). To the right is an expression, E, whose value is calculated and then associated with the program variable from that point on. Program variables can be simple names, such as `x` and `answer` , or array references, such as `a[i]` and `b[5, j]` .

2. A *conditional statement* has the form

   $$\texttt{if}\ T\ \texttt{then}\ S_1\ \texttt{else}\ S_2$$

   Where S_1 and S_2 are, themselves, statements. If the test T holds then statement S_1 is executed; otherwise, statement S_2 is executed;

3. A *repetition statement* has the form

   $$\texttt{while}\ T\ \texttt{do}\,S$$

   Statement S is repeatedly executed so long as the test T remains true.

4. Finally, a *compound statement* has the form

   $$\texttt{begin}\,S_1\,\texttt{;}\,S_2\,\texttt{;}\ \ \ldots\ \ \texttt{;}\,S_n\,\texttt{end}$$

   Statements S_1, S_2, ..., S_n are executed in order.

Figure 1.1 shows an equivalent specification of the *Statement* language. It uses *Backus-Naur* notation, or "BNF," a form often seen in programming manuals. Unlike the description above, Figure 1.1 says nothing about what statements *mean*, only what they look like. BNF only describes what sentences are syntactically correct.

Both descriptions are *self referencing*, meaning that they refer to the language in the process of defining it. Conditional, repetition, and compound statements *contain* statements. Although self reference used in this way may seem natural and intuitive, *not all self-referencing definitions are meaningful.*

$\langle$STMT$\rangle$	::=	$\langle$IVS$\rangle$ `:=` $\langle$TERM$\rangle$	(*assignment*)
	|	`if` $\langle$QFF$\rangle$ `then` $\langle$STMT$\rangle$ `else` $\langle$STMT$\rangle$	(*conditional*)
	|	`while` $\langle$QFF$\rangle$ `do` $\langle$STMT$\rangle$	(*repetition*)
	|	`begin` $\langle$STMT$\rangle$ `;` ... `;` $\langle$STMT$\rangle$ `end`	(*compound*)
$\langle$IVS$\rangle$	::=	$\cdots$	(*identifier*)
$\langle$TERM$\rangle$	::=	$\cdots$	(*expression*)
$\langle$QFF$\rangle$	::=	$\cdots$	(*test*)

Figure 1.1: Partial description of the *Statement* programming language STMT, expressed in Backus-Naur form (Sec. 8.5)

For example, consider the language described by items 1–3 above, leaving out the assignment statement. Can you give an example of a program in that language?

The statement language has no input/output operations. We can talk about the result of a program in terms of the final values of its identifiers. Here is an example of a program in the language of statements:

```
     {A, B ∈ ℕ}
𝒫:   begin
     x := A;
     y := B;
     z := 0;
ℓ1:  while  x ≠ 0 do   { This is the loop ℓ1 }
        begin
        x := x − 1;
ℓ2:     z := z + y
        end
     end
     {z = AB}
```

The program labels, $\mathcal{P}$, ℓ_1 and ℓ_2 are not part of the language—there is no `goto` statement so labels aren't needed—but are used in discussions to refer to points of the program.

Comments written between braces, $\{\cdots\}$, are called *assertions*. For now, comments are optional, but in Chapter **??** they become a formal part of the language, used to reason logically about a program's data state. After some staring perhaps, it should be clear that program $\mathcal{P}$ computes the product of natural numbers A and B, as the comments assert.

Exercises 1.4

1. In the programming language just described, write programs for each of the following specifications.

(a) Assume that program variables x and y have been initialized with values A and B in $\mathbb{N}$. Compute the sum of these values, leaving the result in program variable z, using only the operations of adding or subrtracting 1 to (from) a program variable.

(b) Assume that program variables x and y have been initialized with values A and B in $\mathbb{N}$. Compute the product of x and y using only the operations of addition and subtraction.

(c) Assume that program variables x and y have been initialized with values, A and B in $\mathbb{N}$. Compute the value A^B using only addition and multiplication.

(d) Write a program to compute the quotient, q, and remainder, r of two values initially held in variables x (the dividend) and y, the divisor. Assume that you have only addition and subtraction.

(e) Write a program to compute the *greatest common divisor*, $\gcd(x, y)$, of $A,\ B \in \mathbb{N}$ held in program variables x and y respectively, using only addition and subtraction.

(f) Assume that in addition to '+' and '−' you also have an operation, *half*(v) that divides its operand, v by two. Use this operation to improve the performance of the previous gcd program.

Chapter 2

Relations, Functions

After sets, the most fundamental concept we will use is that of a *relation*.

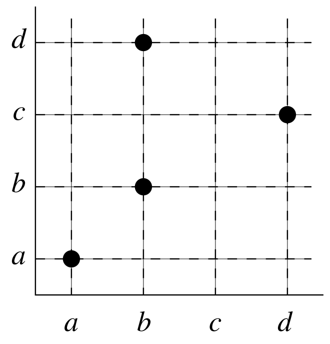
Definition 2.1 *If A and B are sets, then any subset of $A \times B$ may be called a* relation *from* domain *A to* range *B.*

We often simply simply $R \subseteq A \times B$ to say, "R is a relation from A to B." This chapter is devoted to introducing the extensive vocabulary used in classifying and describing relations. We shall begin with the very important class of *functions*, and then discuss a number of other classes.

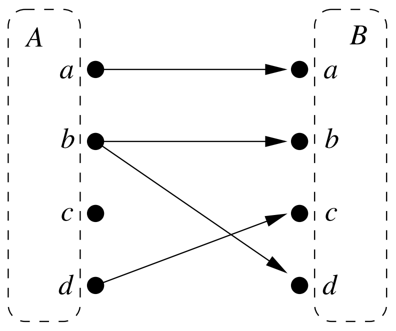
It is often helpful to draw pictures of relations, or *graphs*. In analytic geometry a common form of graph is the *Cartesian product*, after the philosopher/mathematician Réne Descartes who first conceived it. The domain and range are laid out on horizontal and vertical axes. Elements of the relation are shown according to their coordinates on the resulting plane. For example, let the domain A be the set $\{a, b, c, d\}$; let the range B be the set $\{a, b, c, d\}$; and consider the relation from A to B,

$$R = \{(a, a), (b, b), (b, d), (d, c)\}$$

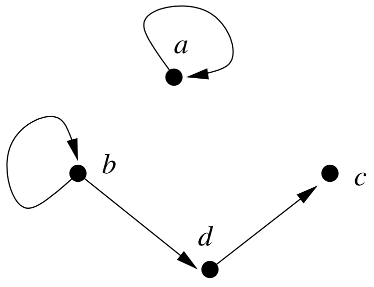
A Cartesian graph of R looks like this:



Another way to represent R, called a *bipartite graph*, is drawn as follows. First, write down all the elements of the domain A in a column (or row). Next, write down all the elements of the range B in another column. Whenever there is an ordered pair, $(x, y) \in R$, draw an arrow from x to y. A bipartite graph of R is shown below:



When the domain and range of a relation are the same set, as is the case here for R, one can draw a *directed graph* representing the relation. In a directed graph, the elements are written down just once. Here is a directed graph of our example R:



The points in a directed graph are called *nodes* or *verticies*. The arrows are called *edges* or *arrows*.

In computer science we often deal with finite, discrete sets. The bipartite and directed graphs of relations on such sets sometimes convey information more clearly than the corresponding Cartesian graphs, and we shall see them often in this book.

2.1 Functions

A common way to think about a function is as "a rule." Sometimes, the rule is specified by a formula; for instance, we might write

$$f(x) = x^2 + 5x + 6$$

to specify a parabolic function. There is the idea of a function as a ”box” that computes a result based on its inputs:

$$\begin{tikzcd} x \arrow[r] & \boxed{f} \arrow[r] & f(x) \end{tikzcd}$$

This notion of a function is close to our notion of a computer program. Too close. The concepts of “function” and “program” differ in some fundamental ways and it is important not to associate them too closely.

Instead, we can think of a function as a particular kind of relation. The property that distinguishes functions from other kinds of relations is that functions associate just one range element for a given domain element.

Definition 2.2 *A* function from X to Y *is a relation* $f \subseteq X \times Y$ *in which, for every* $x \in X$*, there is exactly one* $(x, y) \in f$.

We write

$$f\colon X \to Y$$

to indicate that f is a function from X to Y. If $(x, y) \in f$, we say that y *is the value of* f *at* x*,* and write $f(x) = y$.

Example 2.1 Let $X = \{a, b, c, d\}$; let $Y = \{1, 2, 3, 4\}$; and let

$$f = \{(a, 1), (b, 1), (c, 3), (d, 2)\}$$

The relation $f \subseteq X \times Y$ is a function because each of the possible inputs from domain X is associated with exactly one output. In the bipartite graph of f, we can see that there is just one arrow from each element of the domain.

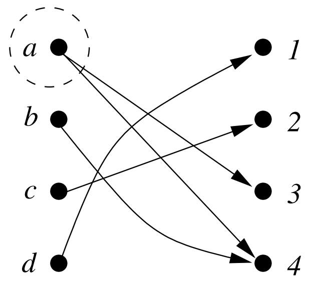
$$\begin{tikzcd} a \arrow[r] & 1 \\ b \arrow[ru] & 2 \\ c \arrow[r] & 3 \\ d \arrow[ruu] & 4 \end{tikzcd}$$

$\square$

Example 2.2 Let sets X and Y be as given in the previous example, and let

$$g = \{(a, 3), (c, 2), (b, 4), (d, 1), (a, 4)\}$$

This relation is not a function because more than one value is associated with a:



It does not make sense to use the notation $g(x) = y$ when g is not a function, for then we would have

$$3 \overset{?}{=} g(a) \overset{?}{=} 4$$

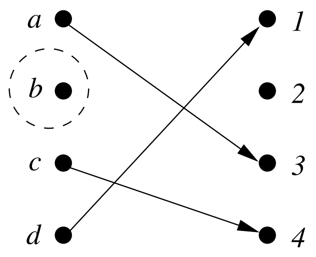
chosen

□

Example 2.3 Let sets X and Y be as given in Example 2.1, and let

$$h = \{(a, 3), (c, 4), (d, 1)\}$$

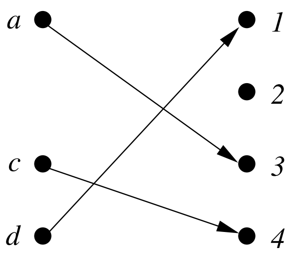
This relation is not a function because h has *no* value for b:



This too is a violation of the conditions of Definition 2.2.

□

Example 2.4 Considered as a relation from $\{a, c, d\}$ to $\{1, 2, 3, 4\}$ the relation $h = \{(a, 3), (c, 2)(d, 1)\}$ *is* a function:



□

In this book, the term "function" always refers to a *well defined* relation, that is, a relation with an ordered pair for every element of its domain. We use the term *partial function* to describe relations such as h in Example 2.3

Definition 2.3 *A relation $f \subseteq X \times Y$ is called a* partial function from X to Y *if for every $x \in X$ there is at most one $y \in Y$ such that $(x, y) \in f$.*

A well defined function is analogous to a "well behaved" program that produces an output for every possible input. A partial function is like a program that diverges—gets "stuck in a loop"—on some of its inputs, producing no output.

We can make a function from any partial function by restricting its domain to the subset of elements for which the function is defined. The following definition give a means of doing that.

Definition 2.4 *Let $f \subseteq X \times Y$ be a partial function; and let $A \subseteq X$ and $B \subseteq Y$. The* image of A under f *is the set*

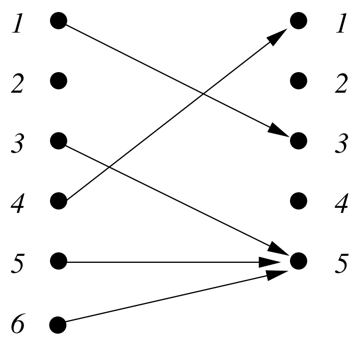
$$\{f(a) \mid a \in A\}$$

and the preimage of B under f *is the set*

$$\{a \mid f(a) = b \text{ for some } b \in B\}$$

For notation, we use fA to denote the image of A under f and $f^{-1}B$ to denote the preimage of B under f.

Example 2.5 Let $X = \{1, 2, 3, 4, 5, 6\}$ and $Y = \{1, 2, 3, 4, 5\}$, and consider the partial function g described by the following bipartite graph:



The following are examples of image and preimage sets:

$$\begin{aligned} gX &= \{1, 3, 5\} \\ g^{-1}Y &= \{1, 3, 4, 5, 6\} \\ g\{4, 5, 6\} &= \{1, 5\} \\ g^{-1}\{2, 4\} &= \emptyset \\ g^{-1}\{4, 5\} &= \{3, 5, 6\} \end{aligned}$$

$\square$

Example 2.6 Let $X = \{a, b, c, d\}$, $Y = \{1, 2, 3, 4\}$, and partial function $h = \{(a, 3), (c, 4), (d, 1)\}$ as in Examples 2.1–2.3. Then h is a function considered as a relation with domain $h^{-1}(Y)$ and range Y.

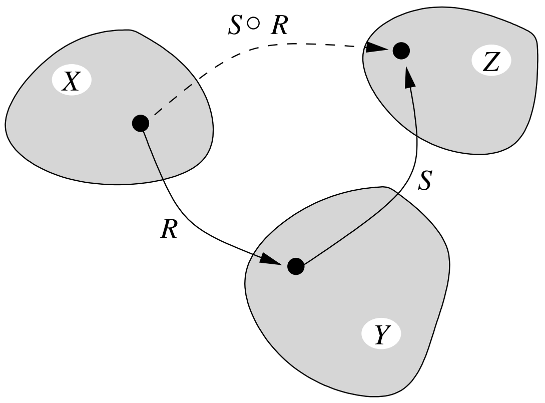
$\square$

2.1.1 Function (and Relation) Composition

Definition 2.5 *Given two relations, $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, the* composition of R and S, *written $S \circ R$, is a relation from X to Z defined by:*

$$S \circ R = \{(x, z) \mid \textit{for some } y \in Y,\ (x, y) \in R \textit{ and } (y, z) \in S\}$$

The picture below illustrates the idea of composition. It defines an edge between elements of X and Z whenever they are both related to a common element in Y:



The next proposition establishes that composing two functions yields a function.

Proposition 2.1 *Let A, B, and C be sets and suppose there are two functions, $f\colon A \to B$ and $g\colon B \to C$. Then $g \circ f$ is a also a function.*

Proof: Let $u \in A$. Since f is a function, there is a unique $f(u) \in B$, such that $(u, f(u)) \in f$. Since g is a function, there is a unique $g(f(u)) \in C$, such that $(f(u), g(f(u))) \in g$. By definition, we have $(u, g(f(u))) \in g \circ f$, and we have shown that this ordered pair is unique with respect to u. Since u was arbitrary, $g \circ f$ is a function.

$\square$

Function composition is usually defined without reference to the more general notion of composition of relations. The typical wording of the definition is

> *Let $f\colon A \to B$ and $g\colon B \to C$. The* composition of g and f *is defined by $(g \circ f)(x) = g(f(x))$.*

Our proof of Proposition 2.1 establishes that $g \circ f$ is "well defined"; that is, $g \circ f$ is actually a function relating exactly on range element to every domain element. Verifying well-definedness is an essential part of the defining process, but it is often left to the reader.

The following definition describes three properties that a function might have.

Definition 2.6 *A function $f\colon X \to Y$ is:*

(a) surjective*, or "onto", iff for each $y \in Y$ there is an $x \in X$ such that $f(x) = y$.*

(b) injective *or "one-to-one", if for every x* and *$x' \in X$, $f(x) = f(x')$ only if $x = x'$.*

(c) bijective *if f is surjective and injective.*

2.1.2 Infix Notation

If $f\colon X \times Y \to Z$ we say f is a *two-place* function taking its first argument from X and its second argument from Y. Such an f is a relation from $X \times Y$ to Z, and a a typical element of f is $((x, y), z)$. Following the notation introduced after Definition 2.2, we should write

$$f((a, b)) = c$$

but it is conventional in mathematics to write

$$f(a, b) = c$$

instead. Similarly, we write $g(a, b, c)$ for the result of a *three-place function*, $g\colon A \times B \times C \to D$, and so on.

Often, the expressions for common two-place functions are even more concise. For example, addition is a two-place function, $+\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$. So, using to our "general purpose" notation for functions, we would write sums as, for example,

$$+(3, 5) = 8$$

Of course we don't write sums that way; instead we write

$$3 + 5 = 8$$

"$+(3, 5)$" is called *prefix* notation and "$3 + 5$" is called *infix* notation. There is also a *postfix* notation, "$(3, 5)+$," that is commonly used in hand-held calculators and some low-level programming languages. Infix is the usual notation for two-place arithmetic operators, and for many other two-place operators as well, such as set operations, concatenation and the logical connectives of in Chapter 3. We use infix in the next definition, which gives names to some special properties of two-place functions.

Definition 2.7 *A two-place function $\odot\colon A \times A \to A$ is said to be*

(a) commutative *iff for all* $x, y \in A$, $x \odot y = y \odot x$;

(b) associative *iff for all* $x, y, z \in A$, $x \odot (y \odot z) = (x \odot y) \odot z$.

(c) *Finally,* $e \in A$ *is an* identity *for* $\odot$ *iff for all* $x \in A$, $x \odot e = e \odot x = x$.

Example 2.7 Integer addition is associative and commutative with zero as an identity. Integer multiplication is associative and commutative with 1 as an identity. Integer subtraction is *neither* commutative *nor* associative. Subtraction does not have an identity as defined; $x - 0 = x$ but $0 - x \neq x$. □

Example 2.8 Concatenation of words is associative and has an identity, ε, but is not commutative. `abc`ˆ`def` $\neq$ `def`ˆ`abc`. □

Exercises 2.1

1. Let $A = \{1, 2\}$ and $B = \{2, 3, 4\}$. Which of the following relations from A to B are functions?

(a) $\{(1, 3),\ (2, 4)\}$
(b) $\{(1, 3),\ (1, 4)\}$
(c) $\{(1, 3),\ (1, 3)\}$
(d) $\{(1, 3),\ (2, 5)\}$
(e) $\{(2, 2),\ (1, 4)\}$

2. Is $\{(1, 2),\ (2, 3)\}$ a function

(a) from $\{(1, 2)\}$ to $\{(2, 3)\}$?
(b) from $\mathbb{N}$ to $\mathbb{N}$?
(c) from $\{1, 2\}$ to $\mathbb{N}$?
(d) from $\{1, 2, 3\}$ to $\{2, 3\}$?
(e) from $\{1, 2, 3\}$ to $\{1, 2, 3\}$?

3. Let $A = \{1, 2\}$ and $B = \{2, 3, 4\}$.

(a) List a relation that is an injective function from A to B.

(b) List a relation that is a surjective function from B to A.

(c) List two bijections from B to B.

4. Let $f\colon A^2 \to A$ be given by the following table:

x	y	$f(x, y)$
1	1	1
1	2	2
2	1	2
2	2	2

Show that f is commutative and associative. What is the identity of f?

5. Let $f\colon A \to B$ and suppose S and T are both subsets of A.

(a) Prove that $f(S \cap T) \subseteq fS \cap fT$.

(b) Take both A and B to be $\{a, b, c\}$. Define a function f and two subsets S and T for which $f(S \cap T) \neq fS \cap fT$

(c) State a condition under which, in general, $f(S \cap T) = fS \cap fT$

6. Let $f\colon A \to B$ and $S \subseteq A$ and $T \subseteq B$ Prove or disprove the following:

(a) $f^{-1}(fS) = S$.

(b) $f(f^{-1}T) = T$.

7. *Prove*: If $f\colon X \times X \to X$, and e and e' are both identities of f, then $e = e'$.

8. *Prove*: If $f\colon A \to B$ and $g\colon B \to C$ are surjections ("onto functions"), then $g \circ f$ is onto.

9. *Prove*: If $f\colon A \to B$ and $g\colon B \to C$ are injections (one-to-one) then $g \circ f$ is an injection.

10. *Prove*: if $f\colon A \to B$ is a bijection, then there exists a bijection from B to A.

2.2 Relations on a Single Set

Let us now consider the important class of relations $R \subseteq A \times A$, in which the domain and range are the same set. The next definition establishes basic terminology for describing relations of this kind.

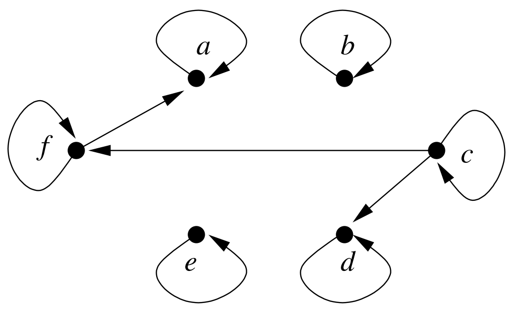
Definition 2.8 *A relation $R \subseteq A \times A$ is*

(a) reflexive *iff for every $a \in A$ $(a, a) \in R$, and* irreflexive *iff for every $a \in A$ $(a, a) \notin R$.*

(b) symmetric *iff $(x, y) \in R$ implies $(y, x) \in R$ and* antisymmetric *iff $(x, y) \in R$ and $(y, x) \in R$ imply $x = y$.*

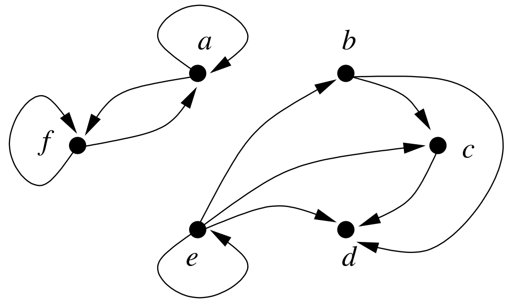
(c) transitive *iff whenever $(x, y) \in R$ and $(y, z) \in R$ it is also the case that $(x, z) \in R$.*

Example 2.9 Let $A = \{a, b, c, d, e, f\}$. The relation R, whose graph is shown below, is reflexive.



It is not symmetric because, for example, $(c, f) \in R$ but $(f, c) \notin R$. □

Example 2.10 Let $A = \{a, b, c, d, e, f\}$, as in the previous example. The relation T whose graph is shown below is transitive, but is neither reflexive, nor symmetric, nor irreflexive, nor antisymmetric.



Saying that a relation is "not symmetric" (or non-symmetric) is not the same as saying it is antisymmetric. Here for example, we have (a, f) and (f, a) in T (so T isn't antisymmetric), but we also have $(e, c) \in T$ while $(c, e) \notin T$ (so T isn't symmetric, either). □

The notion of transitivity suggests that whenever there are two arrows "in sequence," then there is a *single* arrow from the tail of the first to the head of the second. In the relation T of Example 2.10, for example, we have:

$(e, b) \in T$ and $(b, c) \in T$ but also $(e, c) \in T$;
$(b, c) \in T$ and $(c, d) \in T$ but also $(b, d) \in T$;
$(e, c) \in T$ and $(c, d) \in T$ but also $(e, d) \in T$;
and so on.

In general, let R be transitive, and (a_1, a_2), (a_2, a_3), and (a_3, a_4) all be arrows in R. By transitivity, $(a_1, a_3) \in R$, so, applying transitivity to (a_1, a_3) and (a_3, a_4), the arrow $(a_1, a_4) \in R$. This argument can be extended to any chain of arrows. We begin by formalizing the notion of a "chain of arrows."

Definition 2.9 *Let $R \subseteq A \times A$ be a relation. A* path *from a to b in R is a sequence*

$$\langle x_1, x_2, \ldots, x_n \rangle$$

such that

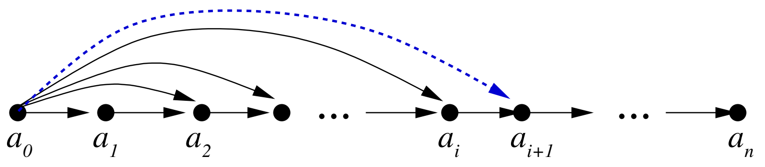
(a) $n \geq 1$
(b) *for each i, $x_i \in A$*
(c) *$x_1 = a$ and $x_n = b$*
(d) *for each i such that $1 \leq i < n$, $(x_i, x_{i+1}) \in R$*

If $a_0 = a_n$, we call the path a cycle. *We say n is the* length *A graph which has no cycles is said to be* acyclic.

Proposition 2.2 *R is transitive iff whenever there is a path from a to b in R, then there is an edge $(a, b) \in R$.*

PROOF: ($\Leftarrow$) Assume that if there is a path from a to b in R, then there is an arrow from a to b in R. We would like to show that R is transitive, that is, if $(a_1, a_2) \in R$ and $(a_2, a_3) \in R$, then $(a_1, a_3) \in R$. If $(a_1, a_2) \in R$ and $(a_2, a_3) \in R$, then $\langle a_1, a_2, a_3 \rangle$ is a path from a_1 to a_3 in R. By the assumption, since there is a path from a_1 to a_3 in R, there is an arrow from a_1 to a_3 in R, which is just what we need to show that R is transitive.

($\Rightarrow$) Assume that $\langle a_1, \ldots, a_n \rangle$ is a path in R, and that R is transitive. We shall show that for each i such that $1 \leq i \leq n$, there is an arrow $(a_0, a_i) \in R$. We shall do this by giving an *algorithm* that, starting with the arrow (a_0, a_1), builds up an arrow (a_0, a_n) by successive applications of transitivity: Imagine we have already built the arrow $(a_0, a_i) \in R$.



Since $\langle a_0, \ldots, a_i, a_{i+1}, \ldots, a_n \rangle$ is a path in R, we know that there is an arrow $(a_i, a_{i+1}) \in R$. Now transitivity requires that there be an arrow $(a_0, a_{i+1}) \in R$. We repeat this "extension" until we reach a_n.

□

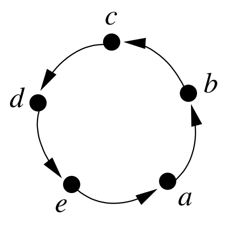
Knowing that all paths are of finite length (Definition 2.9 says this by specifying a length, n) the algorithm certainly demonstrates that the desired edge, (a_1, a_n), exists and it even shows how to determine it. *Constructive* arguments of this form are often used to prove something exists. When a proof is based on an algorithm, one should first ask two questions:

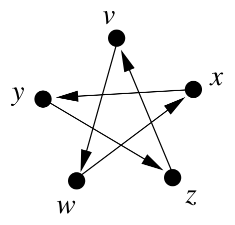
(a) *Is it a definite procedure?* At every step, it must be clear and unambiguous what to do next.

(b) *Does it terminate?* It must be clearly evident that the procedure makes progress toward completion and does not go on forever.

Of course, the algorithm must also achieve its intended purpose—in this case, finding a edge from a_0 to a_n. Proof narratives often focus on the "correctness" aspect, leaving it to the Reader to confirm definiteness and termination.

In computing, we are often more interested in the *structure* of a graph than in details such as the domain of its nodes or in how it is depicted on paper. The two graphs shown below have the same structure, even though their nodes are labeled differently and they are laid out differently:





To have the same structure, an exact correspondence must exist between nodes, and this correspondence must "preserve edges." These qualities are formalized in the next definition.

Definition 2.10 *Two graphs $R \subseteq A \times A$ and $S \subseteq B \times B$ are said to be* isomorphic *iff there exists a bijection $f \colon A \to B$ such that $(a, a') \in R$ iff $(f(a), f(a')) \in S$. The bijection f is called an* isomorphism *between R and S.*

Example 2.11 The bijection represents the correspondence between nodes. In the diagram above, we have $A = \{a, b, c, d, e\}$ and $B = \{v, w, x, y, z\}$. One possible bijection is

$$f = \{(a, x), (b, z), (c, w), (d, y), (e, v)\}$$

Check that there is a resulting correspondence between the edges of the two relations.

□

2.2.1 Attaching Information to Graphs

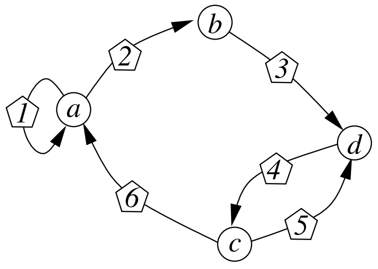
We have already seen graph diagrams that contain information other than the just the graph structure. For instance, the element a node represents is shown next to the node. These instances are annotations that make it easier to interpret the drawing. In many applications, it is desirable to affix information as part of the mathematical representation—as opposed to just depicting it in a drawing. Defining this association is called *labeling*.

Definition 2.11 *Given a relation $R \subset A \times A$, a* labeling *of R is either:*

(a) *an* edge labeling, *$\ell \colon R \to L$, mapping the edges to some set L, or*

(b) *a node labeling, $\ell' \colon A \to L'$, mapping nodes to some set L'.*

Labelings may be depicted in any way that makes clear what the labeling is. The graph below has both node and edge labels, depicted as circles and pentangles, respectively.



$A = \{w, x, y, z\}$, $L = \{a, b, c, d\}$, $L' = \{1, 2, 3, 4, 5, 6\}$ and

$$\begin{array}{rcl} \ell\colon w & \mapsto & a \\ x & \mapsto & b \\ y & \mapsto & c \\ z & \mapsto & d \end{array} \qquad \begin{array}{rcl} \ell'\colon (w,w) & \mapsto & 1 \\ (w,x) & \mapsto & 2 \\ (x,y) & \mapsto & 3 \\ (y,z) & \mapsto & 4 \\ (z,y) & \mapsto & 5 \\ (z,w) & \mapsto & 6 \end{array}$$

In this picture the node's names, w, x, y and z, do not appear, but their labels do. Nevertheless, we will often refer to w as "node a," instead of the more technically correct "the node labeled by a."

Exercises 2.2

1. Let

$$\begin{array}{rcl} A & = & \{a, b, c, d, e\} \\ R & = & \{(a,b), (a,c), (b,a), (c,a), (c,d), (c,e), (d,c), (e,c)\} \end{array}$$

(a) Draw the bipartite graph representation of R.

(b) Draw the directed graph representation of R.

(c) Is R symmetric? reflexive? transitive?

2. Let $R = \{(a,a)\}$, $A = \{a\}$, and $B = \{a, b\}$. Is $R \subseteq A \times A$ reflexive? Is $R \subseteq B \times B$ reflexive? Draw both relations as bipartite graphs and directed graphs.

3. Let $A = \{a\}$, $B = \{a, b\}$.

(a) List all the relations $R \subseteq A \times A$.

(b) List all the relations $R \subseteq B \times B$.

(c) Of the relations in (a) and (b), which are reflexive? Symmetric? Tranansitive?

4. A relation that is not symmetric is said to be *asymmetric*. Draw a directed graph that is asymmetric but not antisymmetric.

5. Draw a directed graph that is symmetric and transitive, but not reflexive.

6. Draw all the directed graphs on a set with two elements. Indicate which of these graphs are isomorphic to one another.

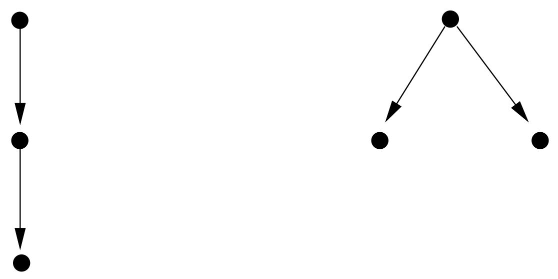
2.3 Trees

Trees are a fundamental structure seen in many areas of computer science. As data structures, trees have the desirable properties for searching and traversal. A number of results about trees are proved in this section. The proofs illustrate a kind of argument, "proof by construction," that is common in computer science.

Definition 2.12 *A* tree *is a finite acyclic directed graph $R \subseteq A \times A$ in which there is one node (called the* root*) with in-degree 0, and every other node has in-degree 1. A node in a tree with out-degree 0 is called a* leaf.

Just as one must include the domain and range in the declaration of relations, and particularly functions, it is necessary to include A in the declaration "$R \subseteq A \times A$ is a tree" (See Exercise 3). One may say, for short, "R is a tree over A."

Example 2.12 There are two distinct trees for a set of three nodes. Here they are:



□

Example 2.13 There are four distinct trees for a set of four nodes. Here they

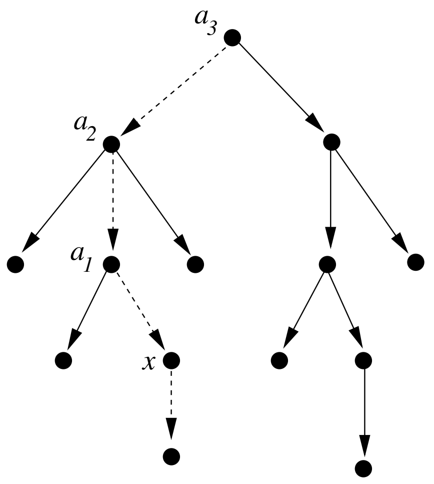
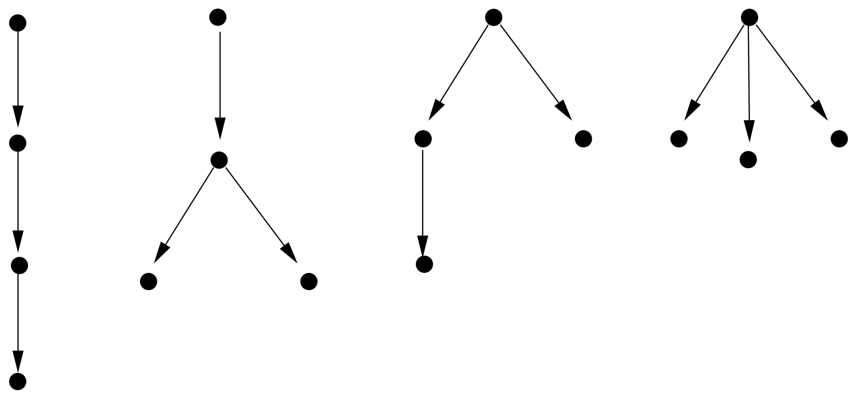


Figure 2.1: Construction of a path from the root of a tree

are:



□

The next result formalizes a property of trees mentioned at the beginning of this section.

Theorem 2.3 *If $R \subseteq A \times A$ is a tree and $x \in A$ is not the root of R, then there is exactly one path from the root to x in R.*

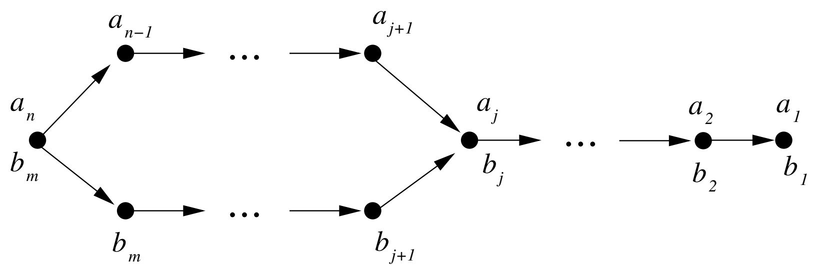
Proof: We shall first construct one path from the root to x, and then show that it is unique. For the first part, see Figure 2.1. Since x is not the root, x has in-degree 1, so there must be a path $\langle a_1, x\rangle$.

Assume we have constructed a path $\langle a_n, a_{n-1}, \ldots, x\rangle$. If a_n is the root, we are done. If a_n is not the root, then the in-degree of a_n is 1, so there must be some node a_n and arrow $(a_{n+1}, a_n) \in R$. Hence, $\langle a_{n+1}, a_n, \ldots, x\rangle$ is a path in R. Now, the length of this path can be no greater than the number of elements in A. For if some node were repeated, then R would have a cycle, contradicting the assumption that R is a tree (Defn. 2.12). Therefore, our path-building procedure must eventually halt; but it can only halt by finding an a_{n+1} which is the root. In other words, the procedure must halt with a path from the root to x.

Now assume that there are two paths

$$\langle a_n, a_{n-1}, \ldots, x\rangle \quad \text{and} \quad \langle b_m, b_{m-1}, \ldots, x\rangle$$

with $a_n = b_m =$ the root. Then at some point the paths must converge: There is an integer j such that $a_{j+1} \neq b_{j+1}$, but $a_j = b_j$, $a_{j-1} = b_{j-1}, \ldots, a_1 = b_1$:



So $(a_{j+1}, a_j) \in R$ and $(b_{j+1}, a_j) \in R$. Therefore, a_j must have in-degree of at least 2, contradicting the assumption that R is a tree (Defn. 2.12). Thus, there cannot be two paths and, by the previous argument, there must be at least one. This concludes the proof.

□

Like the proof of Theorem 2.2, this proof is constructive, describing two algorithms that iterate an unknown number of times before the argument is complete. Recall the discussion after Theorem 2.2 and decide whether the procedures described are definite and terminating. In both cases, what to do next is well determined and iteration is limited either by the finite size of the node set or the finite length of the path.

The proof narrative can be criticized as being "redundant" in the sense that the two parts are very similar and can rather easily be condensed into a single argument combining both existence and uniqueness. (See Exercise 6. However, the argument follows a pattern commonly seen in uniqueness proofs:

existence:	Show that there is *at least one* object with the desired property.
uniqueness:	Show that there is *no more than one* object with the desired property. This is often done by assuming more than one exist and showing that this assumption leads to a contradiction.

This proof strategy is reflected in the phrase, "There is *one* and *only one* x with property $P(x)$," suggesting that there are two things to prove.

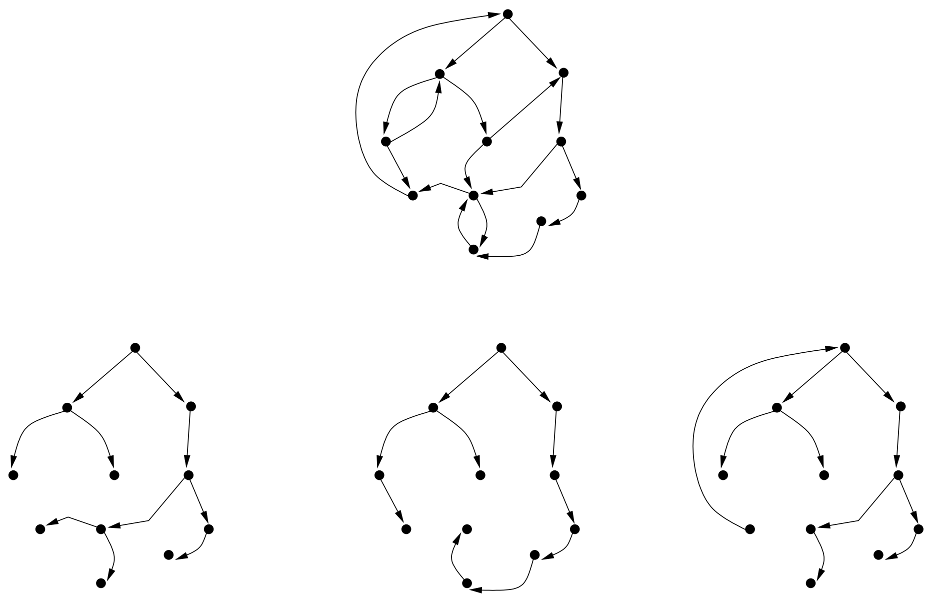


Figure 2.2: A graph and some of its spanning trees.

Because of this unique-path property, it is easy to write programs that visit every node of a tree exactly once. One could use the same algorithms on a general directed graph if one could identify a tree "hidden" in the directed graph Such a hidden tree is called a *spanning tree* of the graph.

Definition 2.13 *If $G \subseteq A \times A$ is a directed graph and if $R \subseteq A \times A$ is a tree and $R \subseteq G$, then we say R is a* spanning tree *of G.*

As Figure 2.2 illustrates, a single graph may have many different spanning trees. If r is the root of a spanning tree R of G, there must be a path in R from r to x for every node in the tree. Since $R \subseteq G$, there must be a path in R from r to x for every node in G. Theorem 2.4 states that this property is all that is needed for G to have a spanning tree.

Definition 2.14 *If $G \subseteq A \times A$ is a* rooted graph *iff there is a node $r \in A$ (the* root*) such that for every $x \in A$ there is a path from r to x in G.*

Theorem 2.4 *Let $G \subseteq A \times A$ by a finite rooted graph with root r. Then G has a spanning tree with root r.*

PROOF: We shall construct a sequence of trees

$$\begin{array}{rcl} R_1 & \subseteq & A_1 \times A_1 \\ R_2 & \subseteq & A_2 \times A_2 \\ & \vdots & \\ R_k & \subseteq & A_k \times A_k \\ & \vdots & \end{array}$$

each with root r and with $R_k \subseteq G$. Each A_k will contain k nodes, so if G has N nodes, $A_N = A$ and R_N will be a spanning tree for G.

First, let $A_1 = \{r\}$ and $R_1 = \emptyset$. (You can check that $R_1 \subseteq A_1 \times A_1$ is a tree.) Now imagine we have built $R_k \subseteq A_k \times A_k$, with $k < N$, and let us construct $R_{k+1} \subseteq A_{k+1} \times A_{k+1}$. Since A_k has k elements, and $k < N$, there must be some $z \in A$ such that $z \notin A_k$. Since G is rooted, let $\langle a_0, a_1, \ldots, a_p \rangle$ be a path from the root $r = a_0$ to $z = a_p$. Since $R_k \subseteq A_k \times A_k$, $r \in A_k$ and $z \notin A_k$, there must be some j such that $a_0, a_1, \ldots, a_j$ all belong to A_k, but $a_{j+1} \notin A_k$.

Set $A_{k+1} = A \cup \{a_{j+1}\}$ and $R_{k+1} = R_k \cup \{(a_j, a_{j+1})\}$.

Now $a_{j+1} \notin A_k$, so (a_j, a_{j+1}) is the only arrow to to a_{j+1}. So a_{j+1} has in-degree 1, and we could not have created a cycle by adding (a_j, a_{j+1}). Furthermore, $(a_j, a_{j+1}) \in G$, so $R_{k+1} \subseteq G$. Last, there is still no arrow ending at r, so r is the root of R_{k+1}. Hence, R_{k+1} has the required properties. Perform the construction N times, and R_N will be the desired spanning tree. $\square$

Figure 2.3 shows the construction of a spanning tree as described in the theorem. Try the construction yourself, using different z's and different paths, to construct a different spanning tree.

Exercises 2.3

1. Draw all the nonisomorphic trees with five vertices.

2. A *binary tree* is a tree in which every nonleaf has an out-degree of two.

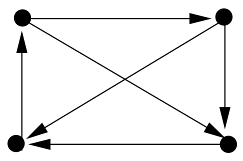
(a) Draw all the distinct (nonisomorphic) binary trees with five nodes.

(b) Draw all the distinct (nonisomorphic) binary trees with six nodes.

(c) Based on you answers to (a) and (b), state a property about binary trees.

3. Let $R = \{(a, b)\}$, $A = \{a, b\}$, $B = \{a, b, c\}$. Is $R \subseteq A \times A$ a tree? Is $R \subseteq B \times B$ a tree?

4. Draw all the spanning trees of the following directed graph:



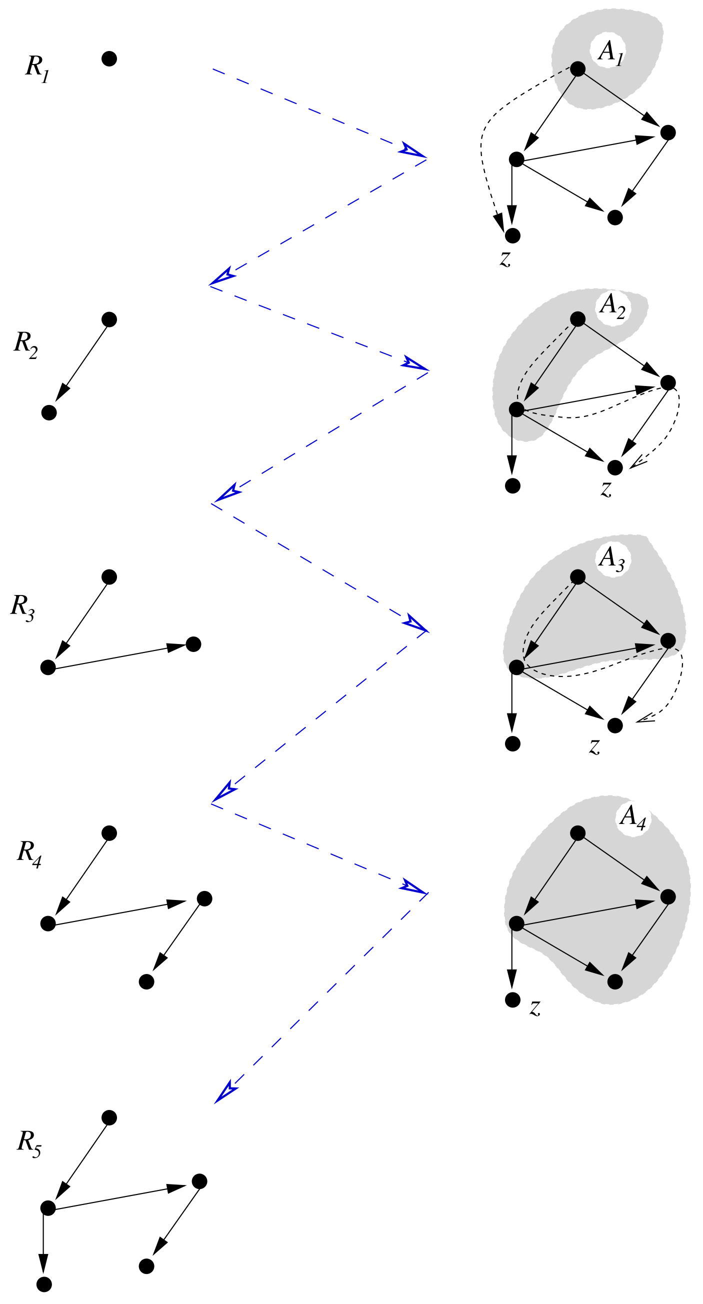


Figure 2.3: Constructing a spanning tree

5. *Prove:* If $A = \{a\}$, there exists exactly one $R \subseteq A \times A$ that is a tree.

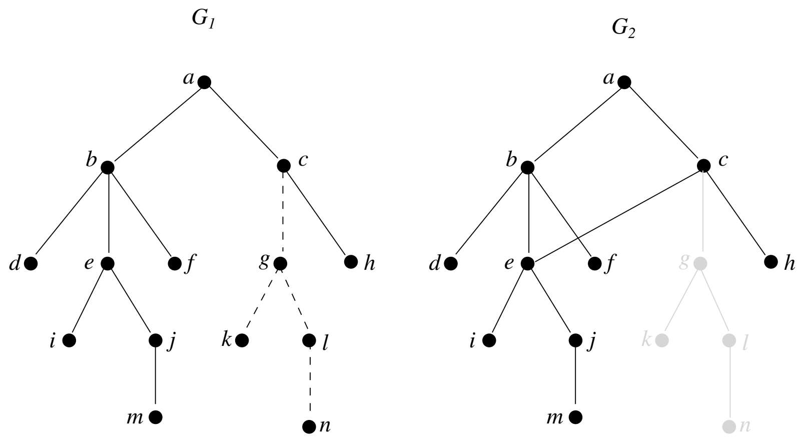
6. Rewrite the proof of Theorem 2.3, combining the two parts of the proof into a single argument. This argument might begin, "Since x is not the root, it has in-degree 1, so (a_1, x) is the only edge leading to x in R; and $\langle a_1, x\rangle$ is the only path of length one ending at x. ...."

2.4 DAGs

Computer data structures often take advantage of the fact that every datum has an address in the computer's memory. If two data structures contain exactly the same information, that information coalesced into a single object whose address may be shared by different access points to the same address. This kind of sharing suggests a kind of graph structure similar to that of a tree but more compact.

Definition 2.15 *A* directed acyclic graph*, or* DAG*, is a rooted graph containing no cyclic paths*

Consider the tree G_1 on the left below. It has isomorphic *sub*trees rooted at e and g. DAG G_2 on the right is obtained by adding the edge (c, e) and removing notes g, k, l and n as well as the edges among them.



The two graphs can be thought of a being "structurally" similar in the sense that whenever there is a path from a to x in G_1, there is a corresponding path in G_2 from a to a node corresponding to x in G_2. For instance, the path

$$\langle a, c, g, l, n\rangle \text{ in } G_1$$

corresponds to

$$\langle a, c, e, j, m\rangle \text{ in } G_1$$

The two paths contain different nodes, so this notion of "similarity" must take into account a correspondence between nodes, as was the case with graph isomorphism defined earlier (Defn. 2.10). This is a weaker correspondence capturing the idea that one graph structure can be embedded in another.

Definition 2.16 *Two directed graphs $R \subseteq A \times A$ and $S \subseteq B \times B$ are said to be* homomorphic *iff there exists a function $h\colon A \to B$ such that if $(a, a') \in R$ then $(h(a), h(a')) \in S$. Such a function h is called a* homomorphism *from A to B.*

Example 2.14 Graphs G_1 and G_2, shown earlier are homomorphic under the mapping h given by

$$\begin{array}{llll}
h\colon a \mapsto a & h\colon e \mapsto e & h\colon i \mapsto i & h\colon m \mapsto m \\
h\colon b \mapsto b & h\colon f \mapsto f & h\colon j \mapsto j & h\colon n \mapsto m \\
h\colon c \mapsto c & h\colon g \mapsto e & h\colon k \mapsto i & \\
h\colon d \mapsto d & h\colon h \mapsto h & h\colon l \mapsto j &
\end{array}$$

□

Exercises 2.4

1. Let $R \subseteq A^2$ $S \subseteq B^2$ $T \subseteq C^2$ and suppose that $f\colon A \to B$ and $g\colon B \to C$ are homomorphisms. Prove that the composition $g \circ f$ is a homomorphism.

2. Some textbooks define a graph homomorphism to be a surjective function:

$$\boxed{\begin{minipage}{0.8\textwidth}\textit{Two directed graphs $R \subseteq A \times A$ and $S \subseteq B \times B$ are said to be homomorphic iff there exists a \underline{surjection} $h\colon A \to B$ such that if $(a, a') \in R$ then $(h(a), h(a')) \in S$.}\end{minipage}}$$

Let us call this kind of homomorphism a *strong homomorphism.* Prove that the composition of two strong homomorphisms is a strong homomorphism.

2.5 Equivalence Relations*

Notions of equivalence are used throughout mathematics A fundamental kind of equivalence is *equality*, between numbers or sets for example. But there are also many ways that we might regard two distinct objects to be equivalent. For example, suppose we have a sack of marbles. We might regard two marbles as equivalent if they are the same size and color. Similarly, we might say two programs are equivalent if they produce the same output for a given input, ignoring other features such as speed, clarity, and so forth.

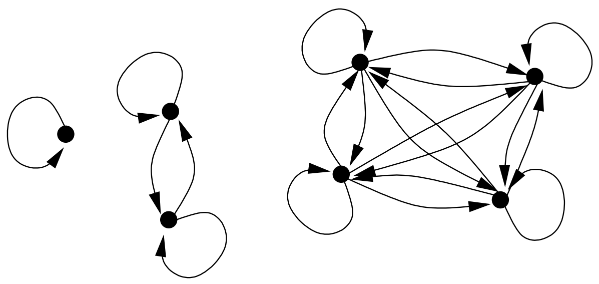
What qualities *must* the notion of equivalence have? In the first place, equivalence is a relation on some set of objects. The following three properties capture the sense of what equivalence means:

- Every object is equivalent to itself.
- Whenever x is equivalent to y, it is also the case that y is equivalent to x.
- Whenever x is equivalent to y and y is equivalent to z, it is also the case that x is equivalent to z.

In other words,

Definition 2.17 *A relation that is reflexive, symmetric, and transitive is called an* equivalence relation*.*

Here is an example of an equivalence relation, depicted as a directed graph:



The picture shows that the nodes are divided into "clusters." Within each cluster, there is an arrow from any node to any node in the same cluster, but between different clusters there are no arrows. It is easy to see why there are no arrows between clusters: If we added an arrow from some node in one cluster to some node in another cluster, then in order to make the relation transitive, we would have to add arrows between all the nodes in the two clusters. The "clusters" are called *equivalence classes.*

Definition 2.18 *Let $R \subseteq A \times A$ be an equivalence relation, and let $a \in A$. The* equivalence class of a under R*, written $[a]_R$, is defined as*

$$\{a \in A \mid (a, a') \in R\}$$

When we can determine R from the context, we omit the "under R" and write just $[a]$. The "clusters" property may be expressed as follows.

Theorem 2.5 *If R is an equivalence relation on A and $a, b \in A$, then*

(a) *if $(a, b) \in R$, then $[a] = [b]$.*

(b) *if $(a, b) \notin R$, then $[a] \cap [b] = \emptyset$.*

Proof: (a) Assume $(a, b) \in R$. We shall show $[b] \subseteq [a]$ and $[a] \subseteq [b]$. To show $[b] \subseteq [a]$, let $x \in [b]$. Then $(b, x) \in R$. Since $(a, b) \in R$ and $(b, x) \in R$, by transitivity, $(a, x) \in R$. Hence, $x \in [a]$. Since we have shown any member of $[b]$ is also a member of $[a]$, $[b] \subseteq [a]$.

To show $[a] \subseteq [b]$, let $y \in [a]$. So $(a, y) \in R$ and since R is a symmetric relation, $(y, a) \in R$. We have assumed that $(a, b) \in R$ and since R is transitive, $(y, b) \in R$. By symmetry again, $(b, y) \in R$; and so $y \in [b]$. Hence $[a] \subseteq [b]$.

(b) We shall show $[a] \cap [b] \neq \emptyset$ implies $(a, b) \in R$. If $[a] \cap [b] \neq \emptyset$. Then there is some z such that $z \in [a]$ and $z \in [b]$. Since $z \in [a]$, we have $(a, c) \in R$, and since $z \in [b]$, we have $(b, c) \in R$ and by symmetry, $(c, b) \in R$. R is a transitive relation, so it follows that $(a, b) \in R$.

□

Corollary 2.6 If $a, b \in A$, then either $[a] = [b]$ or $[a] \cap [b] = \emptyset$.

PROOF: By Theorem 2.5, if $(a, b) \in R$, then $[a] = [b]$; and if $(a, b) \notin R$, then $[a] \cap [b] = \emptyset$.

□

Sometimes, it is preferable to think of a set in terms of its equivalence classes, rather than its individual elements. In programming, for example, this is the difference between a *specification* and an *implementation*. Think of a library of mathematical routines. The user of the library may want a procedure to compute the square root of a number—any number of functionally equivalent routines could be written to do that. The implementer of the library needs to provide one representative of this equivalence class.

Definition 2.19 *If R is an equivalence relation on A, the* quotient set A/R *is* $\{[a]_R \mid a \in A\}$.

Example 2.15 Let $A = \{1, 2, 3\}$ and $R = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3)\}$. Then R is an equivalence class on A and

$$\begin{aligned} [1]_R &= \{1, 2\} \\ [2]_R &= \{1, 2\} \\ [3]_R &= \{3\} \end{aligned}$$

So $A/R = \{\{1, 2\}, \{3\}\}$.

□

Now, A/R is a subset of the power set, $\mathcal{P}(A)$, and so we might ask: which subsets of $\mathcal{P}(A)$ are also quotient sets? That is, which subsets of $\mathcal{P}(A)$ are equal to A/R for some equivalence relation R? The following definition and theorem supply the answers and give us another way to characterize equivalence relations.

Definition 2.20 *Let A be a set. A subset Δ of $\mathcal{P}(A)$ is a* partition *of A iff*

(a) *each $S \in \Delta$ is nonempty, and*

(b) *for each $a \in A$ there is exactly one $S \in \Delta$ such that $a \in S$.*

Theorem 2.7 *A subset Δ of $\mathcal{P}(A)$ is a partition iff $\Delta = A/R$ for some equivalence relation R on A.*

PROOF: ($\Leftarrow$) If R is an equivalence relation, we claim A/R is a partition. If $a \in A$, then $a \in [a]$, so there is some $S \in A/R$ such that $a \in S$. By Corollary 2.6, $[a]$ is the *only* equivalence class containing a. Furthermore, each equivalence class is nonempty. So a/R is a partition.

($\Rightarrow$) The definition of a partition implies that we can define a function, $f\colon A \to \Delta$, mapping each $a \in A$ to the set S of which it is an element. That is,

$$f(a) = S \text{ iff } a \in S$$

Define a relation $R \subseteq A \times A$ as follows

$$R = \{(a,b) \mid f(a) = f(b)\}$$

R is easily seen to be an equivalence relation. We claim $A/R = \Delta$. Let $a \in A$ and $f(a) = S$. Then

$$\begin{aligned} S &= \{x \mid f(x) = S\} \quad (\text{since } x \in S \text{ iff } f(x) = S) \\ &= \{x \mid f(x) = f(a)\} \\ &= [a]_R \end{aligned}$$

So for each $a \in A$, $[a] = f(a) \in \Delta$; hence, $A/R \subseteq \Delta$. Conversely, let $S \in \Delta$. By the definition of partition, S is nonempty, so choose $a \in S$. Hence, $f(a) = S$, and by the previous argument, $S = [a]_R$. So $\Delta \subseteq A/R$. □

Exercises 2.5

1. List A/R for each of the following equivalence relations:

(a) $A = \{x \in \mathbb{N} \mid 1 \le x \le 6\}$
$R = \{(x,y) \mid (x-y) \text{ is evenly divisible by } 3\}$

(b) $A = \{x \in \mathbb{N} \mid 1 \le x \le 8\}$
$R = \{(x,y) \mid x = 2^i k \text{ and } y = 2^j k \text{ for odd } k\}$

(c) $A = \{x \in \mathbb{N} \mid 1 \le x \le 24\}$
$R = \{(x,y) \mid x = 2^i 3^j k \text{ and } y = 2^{i'} 3^{j'} k' \text{ and } i + j = i' + j'$
$\text{and } k,\, k' \text{ are not evenly divisible by 2 or 3}\}$

2. If A and B are sets and $f\colon A \to B$, define the *kernal* of f [denoted $\mathrm{Ker}(f)$] to be $\{(x,y) \mid x, y \in A \text{ and } f(x) = f(y)\}$. Prove that for any f, $\mathrm{Ker}(f)$ is an equivalence relation.

3. *Prove:* Suppose R is an equivalence relation on A. Define a set B and a function $f\colon A \to B$ such than $R = \mathrm{Ker}(f)$.

2.6 Partial Orders*

Set containment and numeric inequality, are examples of a class of relations called *partial orders*. These relations are of great importance in the theory of computer science.

Definition 2.21 *A relation that is reflexive, antisymmetric and transitive is called a* partial order*. If the condition of antisymmetry is removed, the relation is called a* preorder*.*

Example 2.16 Set containment is a partial order. The proofs are left as exercises:

- *Reflexivity.* If S is any set, then $S \subseteq S$.
- *Antisymmetry.* If $S \subseteq T$ and $S \neq T$, then it is not the case that $T \subseteq S$.
- *Transitivity.* If $S \subseteq T$ and $T \subseteq U$ then $S \subseteq U$.

□

A special case of partial order is one in which all the elements are related to all others.

Definition 2.22 *A* total order *is a partial order $R \subseteq A \times A$ with the additional property that for all $x, y \in A$, either $(x, y) \in R$ or $(y, x) \in R$.*

Example 2.17 $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$ are assumed to be totally ordered by the relation '$\leq$'.

□

Example 2.18 A tree is not a partial order because trees are neither reflective nor transitive. There are many times when one wants to "extend" the graph of a tree to make it into a partial order, incorporating concepts like "node m is a descendent of node n." Extending a tree T (or any graph for that matter) to a partial order is conceptually straightforward:

(a) To make $T \subseteq A \times A$ reflexive, add self-edges to every $a \in A$. The graph $T^r = T \cup \{(a, a) \mid a \in A\}$ is called the *reflexive closure of* T.

(b) To make T transitive, add an edge from (a, b) to T whenever there is a path $\langle a, n_1, \ldots, n_{k-1}, b\rangle$ in T. The resulting graph, call it T^*, is called the *transitive closure of* T^r.

□

Although it is intuitively clear what it is, writing down a definition of T^* is thought provoking. T^* could be defined as

$$T^* = T^r \cup \{(a, b) \mid \text{there is a path from } a \text{ to } b \text{ in } T^*\}$$

In this definition T^* is being defined in terms of *itself*, and we need to consider whether this equation is meaningful. A self referencing definitions are not always valid or even meaningful. This point is discussed further in Chapter 7.

Example 2.19 Define the relation $R \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ on ordered pairs as follows:

$((n, m), (k, l)) \in R$ iff

(a) $n \leq k$ or

(b) $n = k$ and $m \leq l$.

That that R is a partial order follows from the fact that '$\leq$' is a partial order. It is reflexive because, for all $n, m \in \mathbb{N}$, $((n, m), (n, m)) \in R$; and if $((n, m), (k, l)) \in R$ and $((k, l), (u, v)) \in R$, so is $((n, m), (u, v)) \in R$. To prove transitivity, there are several cases to consider, specifically,

- $n = k = u$
- $n = k \leq u$
- $n \leq k = u$
- $n \leq k \leq u$

Check that in each case, transitivity holds. Because this ordering is like an "alphabetical" listing, it is called *the lexigraphic ordering* and is often denoted by the symbol $\leq^L$. Notice that even though '$\leq$' is a total order, '$\leq_L$' is not, and that even when the underlying ordering is not total, the lexigraphic ordering is still well defined.

□

Example 2.20 Let A be an alphabet and consider the language A^* of all words built from letters in A, including ε. If $u, v \in A^*$ the u is called a *prefix* of v if there exists a word w such that $u\hat{\ }w = v$. Check that the prefix relation is a partial order.

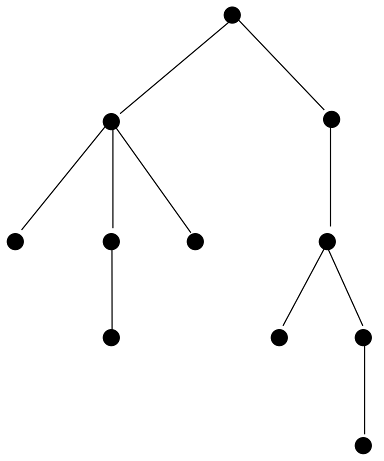
□

Exercises 2.6

1. In the lexigraphic ordering of $\mathbb{N}^2$, give an example of two elements that are not related.

2. In the prefix ordering of A^*, give an example of two elements that are not related.

3. Defintion 2.21 states that a *partial order* is a reflexive, anti-symmetric, transitive relation. Add the edges needed to extend this tree to a partial order.



Chapter 3

Propositional Logic and Boolean Algebra

3.1 Propositions and Truth Tables

A *proposition* is a statement of fact, a sentence to which a value of *true* or *false* can be assigned. Compound propositions are built from simpler propositions using *logical connectives*, such as "and," "or," and "implies." A *propositional formula* is an expression involving simple propositions and logical connectives. Suppose that P and Q stand for propositional formulas. Then the following are also propositional formulas:

$\neg P$	*negation*	$P \Rightarrow Q$	*implication*
$P \land Q$	*conjunction*	$P \Leftrightarrow Q$	*coincidence*
$P \lor Q$	*disjunction*	$P \not\Leftrightarrow Q$	*exclusive-or*

Figure 3.1 shows some of the ways these connectives are expressed in English. The following definition tells what the connectives mean.

Definition 3.1 *The tables below define how the propositional connectives are interpreted.*

P	$\neg P$
F	T
T	F

P	Q	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$	$P \not\Leftrightarrow Q$
F	F	F	F	T	T	F
F	T	F	T	T	F	T
T	F	F	T	F	F	T
T	T	T	T	T	T	F

3.1.1 Implication*

Implication, $P \Rightarrow Q$, sometimes becomes confusing when considered in isolation. Since explaining it may simply compound the confusion, you may wish to defer

$$\neg P \qquad \left\{ \begin{array}{l} \text{It is not the case that } P. \\ P \text{ does not hold.} \\ \text{Not } P. \end{array} \right.$$

$$P \wedge Q \qquad \left\{ \begin{array}{l} P \text{ and } Q. \\ P \text{ but } Q. \end{array} \right.$$

$$P \vee Q \qquad \left\{ \begin{array}{l} P \text{ or } Q. \\ \text{either } P \text{ or } Q \text{ or both} \\ \text{at least one of } P \text{ and } Q \end{array} \right.$$

$$P \Rightarrow Q \qquad \left\{ \begin{array}{l} P \text{ implies } Q. \\ \text{if } P \text{ then } Q. \\ Q \text{ whenever } P. \\ Q \text{ if } P. \\ P \text{ only if } Q. \\ P \text{ is sufficient for } Q. \\ Q \text{ is necessary for } P. \\ Q \text{ follows from } P. \end{array} \right.$$

$$P \Leftrightarrow Q \qquad \left\{ \begin{array}{l} P \text{ if and only if } Q. \\ P \text{ iff } Q. \\ P \text{ exactly when } Q. \\ P \text{ is necessary and sufficient for } Q. \\ \text{Whenever } P \text{ then } Q \text{ and conversely.} \end{array} \right.$$

$$P \not\Leftrightarrow Q \qquad \left\{ \begin{array}{l} \text{Either } P \text{ or } Q \text{ but not both.} \\ \text{Exactly one of } P \text{ and } Q. \\ P \text{ exclusive-or } Q. \end{array} \right.$$

Figure 3.1: The logical connectives and some corresponding English utterances.

reading this section until a question like *"What does $\Rightarrow$ really <u>mean</u>?"* comes to mind and motivates you to read about it.

We are not accustomed to thinking about what "P implies Q" should mean when the *antecedent* P is known to be false or when the *consequent* Q is known to be true. In most mathematical arguments, there is a connection between the two. P must be *used* to carry the argument through. The following examples illustrate why the definition of implication is natural.

Example 3.1 **Proposition** *If A is any set, then $\emptyset \subseteq A$.*

PROOF: According to Definition 1.4, $\emptyset \subset A$ is true provided, "every element of $\emptyset$ is also an element of A." This means that the statement $x \in \emptyset \Rightarrow x \in A$ must be valid, no matter what element is chosen for x. But no matter what x is, "$x \in \emptyset$" is a false statement. In other words, "$\emptyset \subseteq A$" logically reduces to $F \Rightarrow x \in A$, and by Definition 3.1, this proposition is *true* whether or not $x \in A$.

□

Another way to put it is that no choice of x exists that can be used falsify "$x \in \emptyset \Rightarrow x \in A$." It cannot be false, so it must be true. We say that the proposition holds *vacuously*, since the antecedent is *logically false*. □

Example 3.2 **Proposition** *If A is any set, then $A \subseteq A$.*

PROOF: According to Definition 1.4, $A \subseteq A$ means that the statement $x \in A \Rightarrow x \in A$ must be valid no matter what x is. But if "$x \in A$" is true, then the statement reduces to $T \Rightarrow T$, and if $x \notin A$ we have $F \Rightarrow F$. In either case, the proposition is *true*. □

We say that the proposition is *tautologically valid* because it reduces to a purely logical trueism. See Definition 3.2 later in this chapter. □

Example 3.3 **Proposition** *For all $n, m \in \mathbb{Z}$, if $a > 0$ and $b > 0$ then $(a+b)^1 \geq a^1 + b^1$.*

PROOF: By definition, raising any number to the "first power" yields the same number. In other words, for any $z \in \mathbb{Z}$, $z^1 = z$. Thus,

$$(a + b)^1 = a + b = a^1 + b^1$$

so it follows immediately that $(a + b)^1 \geq a^1 + b^1$, as desired. □

The antecedent "$a > 0$ and $b > 0$" is irrelevant, the truth of the proposition does not depend on whether or not a and b is positive. (Had the proposition been $(a + b)^2 \geq a^2 + b^2$, the antecedent *would* be relevant.)

We say that this proposition holds *trivially*, that is, the consequent holds independently of the antecedent. □

A good way to think about $P \Rightarrow Q$ is that it says, *"either P is false or Q is true, or possibly both."* Or, *"it is never the case that Q is true and P is false."* Or, *"Q (is true) only if P (is also true)."*

Another source of confusion lies in the multiple roles implication plays in mathematical discourse. Proposition 3.1, later in this chapter, uses implication ("if and only if") in its statement. It is a statement *about.* And its proof is a chain of implications.

3.2 Truth Tables

Definition 3.1 gives us the means to evaluate complex propositions. We do so by first evaluating the innermost terms and then working outward. We keep track of intermediate results in a *truth table*. Simple examples of truth tables are used in Definition 3.1.

Example 3.4 *Evaluate the formula $(P \vee R) \Rightarrow Q$.*

Solution: A truth table for this formula includes one row for each combination of truth values that might be assigned to its sub-formulas. In this case there are eight possibilities. To the right is the evaluation of the formula. Subterms $P \vee R$ (1) and Q (2) are evaluated and then the '$\Rightarrow$' (3).

P	Q	R	$(P \vee R)$	$\Rightarrow$	Q	
F	F	F	F	T	F	
F	F	T	T	F	F	$\leftarrow$
F	T	F	F	T	T	
F	T	T	T	T	T	
T	F	F	T	F	F	$\leftarrow$
T	F	T	T	F	F	$\leftarrow$
T	T	F	T	T	T	
T	T	T	T	T	T	
			(1)	(3)	(2)	

According to the table, there are three cases in which the proposition $(P \vee R) \Rightarrow Q$ is false.

$\square$

Example 3.5 Truth tables are sometimes used to analyze "story problems." Consider the following example:

Dick, Jane, and Sally are working together on a programming project. Dick says, "Sally's routine is correct; but my routine is correct only if Jane's is." Sally says, "If my routine has a bug, so does Dick's; but my routine is correct." Jane says, "Either Dick's routine has a bug or Sally's does; but not both."

Assuming all three are telling the truth, whose routine has a bug? Whose is correct? Assuming all the routines are correct, who's not telling the truth?

SOLUTION: Let us identify the atomic propositions. Define D, J and S as follows:[†]

$$\begin{aligned} D &\equiv \text{“Dick's routine is correct.”} \\ J &\equiv \text{“Jane's routine is correct.”} \\ S &\equiv \text{“Sally's routine is correct.”} \end{aligned}$$

The assertions made by the three programmers are

$$\begin{aligned} \text{Dick:}\quad & S \wedge (D \Rightarrow J) \\ \text{Jane:}\quad & D \not\Leftrightarrow S \\ \text{Sally:}\quad & (\neg S \Rightarrow \neg D) \wedge S \end{aligned}$$

and we are interested in the truth of the proposition $D \wedge J \wedge S$. Here is a truth table:

D J S	$S \wedge (D \Rightarrow J)$	$D \not\Leftrightarrow S$	$(\neg S \Rightarrow \neg D) \wedge S$	
F F F	F	F	F	
F F T	T	T	T	$\leftarrow$
F T F	F	F	F	
F T T	T	T	T	$\leftarrow$
T F F	F	T	F	
T F T	F	F	T	
T T F	F	T	F	
T T T	T	F	T	

For both cases in which Dick's, Jane's, and Sally's statements are true, D is false and S is true. Thus, we can conclude that, Dick's routine has a bug and Sally's does not, provided all three programmers are telling the truth. We cannot draw any conclusion about Jane's routine. The last row of the truth table is the case where all three routines are correct; and in that row, Jane's statement is false.

□

3.2.1 Logical Equivalence

Propositions may be characterized and compared using truth tables. Our study of mathematical reasoning in later chapters involves truth-table analysis of the assertions made in formal proofs. The next two definitions provide a basic vocabulary for classifying propositions.

Definition 3.2 *A proposition is called a* tautology *when all rows of its truth table evaluate to T. A proposition is called a* contradiction *when all rows of its truth table evaluate to F. A proposition which is neither a tautology nor a contradiction is called a* contingency*.*

[†] Recall the remark about linguistic identification on page 3. A triple-equals sign is used when names are assigned to formulas.

Definition 3.3 *Two propositions, P and Q, are said to be* logically equivalent *when their truth tables are identical. Logical equivalence is written*

$$P \mathbf{\ eq\ } Q$$

Logical equivalence and logical coincidence are very similar concepts. The need to make a distinction between ' **eq** ' and '$\Leftrightarrow$' will be made clearer in Chapter **??** on formal logic.

Proposition 3.1 *P **eq** Q iff $P \Leftrightarrow Q$ is a tautology.*

How many logical connectives do we need? As the following proposition says, all the connectives we have defined can be implemented using only negation and disjunction.

Proposition 3.2 *The following pairs of formulas are logically equivalent:*

$$\begin{array}{llcc}
(a) & P \Rightarrow Q & \textit{and} & (\neg P) \vee Q \\
(b) & P \wedge Q & \textit{and} & \neg((\neg P) \vee (\neg Q)) \\
(c) & P \Leftrightarrow Q & \textit{and} & (P \Rightarrow Q) \wedge (Q \Rightarrow P) \\
(d) & P \not\Leftrightarrow Q & \textit{and} & \neg(P \Leftrightarrow Q)
\end{array}$$

PROOF: In each case logical equivalence is established by a comparison of truth tables, as specified in Definition 3.3. The tables for part (a) are shown below and the rest of the proof is left as an exercise.

P	Q	P	$\Rightarrow$	Q	$\neg P$	$\vee$	Q
F	F	F	T	F	F	T	F
F	T	F	T	T	F	T	T
T	F	T	F	F	T	F	F
T	T	T	T	T	T	T	T

□

Other sufficient sets of connectives are developed as exercises. There is much more to say about propositions. But now we have enough information about them to consider the next topic of this chapter.

Exercises 3.2

1. Let P stand for the proposition "Sue says it." Let Q stand for the proposition "Sam saw it." Let R stand for the proposition "Sid did it." Express the following sentences as formulas involving the logical connectives. If there is more than one way to translate a sentence, use truth tables to explain any differences in the meaning among these translations.

(a) Sid did it, Sam saw it, and Sue says it.

(b) If Sid did it, Sam saw it.

(c) Sid did it only if Sam saw it.

(d) Sue says it only if Sid did it, and Sam saw it.

(e) If Sue says it implies Sam saw it, Sid did it.

2. Definition 3.1 gives the meaning of five logical operations of two arguments. How many distinct logical connectives of two arguments are there?

3. Consider the logical operation defined below:

P	Q	$P \downarrow Q$
F	F	T
F	T	T
T	F	T
T	T	F

Show that '↓' can be used to implement (in the sense of Prop. 3.2) all of the operations of Definition 3.1.

4. Determine another logical operation, different than ↓, which can be used to implement all of the operations of Definition 3.1.

3.3 Boolean Algebra

Digital computers are based on electrical systems in which there are just two voltage values. (*voltage* is a measure of electrical force). All the components of a digital system are carefully designed to produce and respond to just these two levels. Mathematically, the binary values of a digital system are represented as a two-element set of binary digits, or *bits*, $\{0, 1\}$. The basic operations on bits are defined below.

Definition 3.4 *The bit-operations of* inversion*,* addition*, and* multiplication *are given by the following tables.*

$\overline{x}$	
0	1
1	0

$\cdot$	0	1
0	0	0
1	0	1

$+$	0	1
0	0	1
1	1	1

The multiplication sign is dropped where possible, so that the expression

$$(\overline{x} \cdot y) + \overline{(x \cdot \overline{y})}$$

is usually written

$$\overline{x}\, y + \overline{x\, \overline{y}}$$

There is a correspondence between logical connectives and boolean operations

'='	$\mapsto$	*logical equivalence*	$\overline{X}$	$\mapsto$	$\neg X$
1	$\mapsto$	*true*	$X \cdot Y$	$\mapsto$	$X \wedge Y$
0	$\mapsto$	*false*	$X + Y$	$\mapsto$	$X \vee Y$

As we shall see in the next section, this is not the only correspondence possible, but it is nevertheless valid to manipulate propositional terms using the algebraic laws defined next.

Proposition 3.3 *Assume variables x, y, and z range over bits, and let the operations of negation, addition, and multiplication be as defined in Definition 3.4. These operations obey the following algebraic identities.*

Boolean Identities		
Negation	$\overline{\overline{x}} = x$	
Identity	$0 + x = x$	$1 \cdot x = x$
Dominance	$1 + x = 1$	$0 \cdot x = 0$
Idempotence	$x + x = x$	$x \cdot x = x$
Cancellation	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
Commutativity	$x + y = y + x$	$x \cdot y = y \cdot x$
Associativity	$x + (y + z) = (x + y) + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Distributivity	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	$x + (y \cdot z) = (x + y) \cdot (x + z)$
DeMorgan	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	$\overline{(x \cdot y)} = \overline{x} + \overline{y}$

Proof: Each of the laws can be verified by comparing boolean truth tables according to Definition 3.4. □

Definitions 3.4 and 3.3 form a system in which we can reason about equality. As the exercises at the end of this section illustrate, we can generalize this *algebra*, or system of identities, to structures other than $\{0, 1\}$.

Definition 3.5 *A set B containing distinguished elements 1 and 0, and having operations '$\cdot$ ', '$+$', and '$\overline{}$' which satisfy the laws of Proposition 3.3 for x, y, and z ranging over B, is called a* Boolean Algebra*.*

Example 3.6 Use the boolean identities to show that $a(a + b) = a$.

Solution: We shall show this by performing a derivation starting from the left-hand side.

$$\begin{aligned} a(a+b) &= (a+0)(a+b) && (\textit{identity}) \\ &= a+0b && (\textit{distributivity}) \\ &= a+0 && (\textit{dominance}) \\ &= a && (\textit{identity}) \end{aligned}$$

□

Example 3.7 Use the boolean identities to show that $a + ab = a$.

Solution: Here is the derivation:

$$\begin{aligned} a+ab &= a1+ab && (\textit{identity}) \\ &= a(1+b) && (\textit{distributivity}) \\ &= a1 && (\textit{dominance}) \\ &= a && (\textit{identity}) \end{aligned}$$

□

3.3.1 Duality

Compare the derivations in Examples 3.6 and 3.7, just above. The same laws are applied in the same order, the difference being that addition and mulitplication are interchanged, as are 1s and 0s. Looking at the table in Proposition 3.3, it may be evident that one can always do this transformation.

The laws dealing with boolean addition and multiplication come in pairs. For every identity which holds for addition, there is a corresponding identity for multiplication, and conversely. This property is called *duality.*

We have already suggested a correlation between the boolean values and operations, and truth values with logical operations. For if we think of the bit 0 as meaning *false* and the bit 1 as meaning *true,* then inversion, addition, and multiplication, implement negation, disjunction, and conjunction, respectively. Compare the following tables with those in Definition 3.1:

$\neg x$	
F	T
T	F

$\wedge$	F	T
F	F	F
T	F	T

$\vee$	F	T
F	F	T
T	T	T

But there is another way to associate truth values with bits. Let bit 0 represent *true* and bit 1 represent *false.* In this case, bit addition implements logical conjunction and bit multiplication implements disjunction:

$\neg x$	
T	F
F	T

$\vee$	T	F
T	T	T
F	T	F

$\wedge$	T	F
T	T	F
F	F	F

Associating *true* with bit 1 is called the *positive logic interpretation*. and associating *false* with 1 is called the *negative logic interpretation*. Digital engineers use both interpretations when implementing digital "logic."

Exercises 3.3

1. Using boolean truth tables the Distributivity and DeMorgan identities in Proposition 3.3

2. Reduce the following boolean expressions to simpler terms

(a) $xy + (x + y)\overline{z} + y$

(b) $x + y + \overline{(\overline{x} + y + z)}$

(c) $yz + wx + z + [wz(xy + wz)]$

3. Define $x \oplus y$ to be $x\overline{y} + \overline{x}y$. Use boolean algebra to prove

(a) $x \oplus y = \overline{x} \oplus \overline{y}$

(b) $x(y \oplus z) = xy \oplus xz$

(c) $\overline{(x \oplus y)} = \overline{x} \oplus y$

4. Let $A = \{a, b, c\}$ and define the following correspondence for $\mathcal{P}(A)$:

$$\begin{array}{rcl} 1 & \mapsto & A \\ 0 & \mapsto & \emptyset \\ \overline{X} & \mapsto & A \setminus X \\ X \cdot Y & \mapsto & X \cap Y \\ X + Y & \mapsto & X \cup Y \end{array}$$

Show that this correspondence forms a Boolean algebra, according do Definition 3.5.

5. Let D be the set of numbers that divide 30, $D = \{1, 2, 3, 5, 6, 10, 15, 30\}$, and define the following correspondence.

$$\begin{array}{rcl} 1 & \mapsto & 30 \\ 0 & \mapsto & 1 \\ \overline{x} & \mapsto & 30 \div x \\ x \cdot y & \mapsto & \text{the greatest common divisor of } x \text{ and } y \\ x + y & \mapsto & \text{the least common multiple of } x \text{ and } y \end{array}$$

Show that this correspondence forms a Boolean algebra.

6. Show that the connectives '$\wedge$', '$\vee$', and '$\neg$' form a Boolean algebra under a notion of equality that says, $P = Q$ iff P is logically equivalent to Q.

3.4 Normal Forms

Consider the truth table, given below, for the proposition $A \Leftrightarrow B$ where

$$\begin{array}{lcl} A & \equiv & (P \Rightarrow Q) \wedge (Q \Rightarrow R) \\ B & \equiv & (P \Rightarrow R) \end{array}$$

			A			$\Leftrightarrow$	B
P	Q	R	$(P \Rightarrow Q)$	$\wedge$	$(Q \Rightarrow R)$	$\Leftrightarrow$	$(P \Rightarrow R)$
F	F	F	T	$T^\bullet$	T	T	T
F	F	T	T	$T^\bullet$	T	T	T
F	T	F	T	F	F	F	T
F	T	T	T	$T^\bullet$	T	T	T
T	F	F	F	F	T	T	F
T	F	T	F	F	T	F	T
T	T	F	T	F	F	T	F
T	T	T	T	$T^\bullet$	T	T	T

Comparing the truth tables for A and B one can see that they are *not* logically equivelent.

Look now at the truth table for formula A. It is true in four cases and false in the other four. We can construct a formula describing just the *true* cases by recording the truth values of P, Q and R.

A is true just when

$P = F$, $Q = F$ and $R = F$, or

$P = F$, $Q = F$ and $R = T$, or

$P = F$, $Q = T$ and $R = T$, or

$P = T$, $Q = T$ and $R = T$.

In each case, a formula can be written singling out exactly that case

A is true just when

$(\neg P) \wedge (\neg Q) \wedge (\neg R)$ is true, or

$(\neg P) \wedge (\neg Q) \wedge R$ is true, or

$(\neg P) \wedge Q \wedge R$ is true, or

$P \wedge Q \wedge R$ is true.

Equivalently,

$$A \ \mathbf{eq}\ (\neg P \wedge \neg Q \wedge \neg R) \ \vee\ (\neg P \wedge \neg Q \wedge R) \ \vee\ (\neg P \wedge Q \wedge R) \ \vee\ (P \wedge Q \wedge R)$$

Similarly,

$$\begin{array}{ll} B \ \mathbf{eq} & (\neg P \wedge \neg Q \wedge \neg R) \ \vee\ (\neg P \wedge \neg Q \wedge R) \ \vee\ (\neg P \wedge Q \wedge \neg R) \\ & \vee\ (\neg P \wedge Q \wedge R) \ \vee\ (P \wedge \neg Q \wedge R) \ \vee\ (P \wedge Q \wedge R) \end{array}$$

One can construct such a formula from any truth table. In essence, it is just a way of describing the truth table. The result is always disjunction of *and-clauses*, each of which contains every variable just once in either a positive or negated instance.

The formula thus derived is called the *disjunctive normal form* (*DNF*) of the original expression, A in this case.

Since every proposition has a logically equivalent DNF, we have shown that for any logical expression, there is an equivalent one expressed in terms of '$\wedge$', '$\vee$' and '$\neg$', as suggested by Proposition 3.2.

We have now seen two essentially equivalent ways to represent propositions in a way a way that makes them easier to analyze or compare: truth tables and DNFs. Both of them suggest computer encodings that could be used in automating the analyses.

(a) An array of 1s and 0s could be used to represent the truth table. For the proposition A

1	1	0	1	0	0	0	1

(b) A list of 3-bit quantities—a bit for each variable, one for each clause in the DNF. For the proposition A the list would be

```
(000 001 011 111)
```

In both cases, the encodings contain the essential information characterizing A. However, to interpret the encodings, one needs additional information about *how many* variables are used in the formula and *in what order* those variables are used in developing the truth table.

3.4.1 Decision Diagrams*

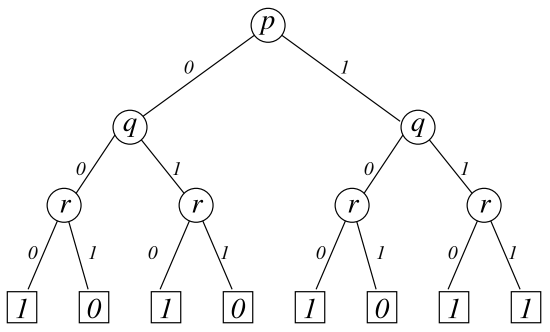
The boolean expression $E \equiv \overline{p}\,\overline{r} + \overline{q} + r$ has the truth table

p	q	r	E
0	0	0	1
0	0	0	0
0	1	0	1
0	1	0	0
1	0	0	1
1	0	0	0
1	1	0	1
1	1	0	1

Section 3.4 introduced a way to represent this truth table in disjunctive normal form written as a propositiional term. Written a boolean term under a positive-logic interpretation, the analogous form is called *sum-of-products form*.

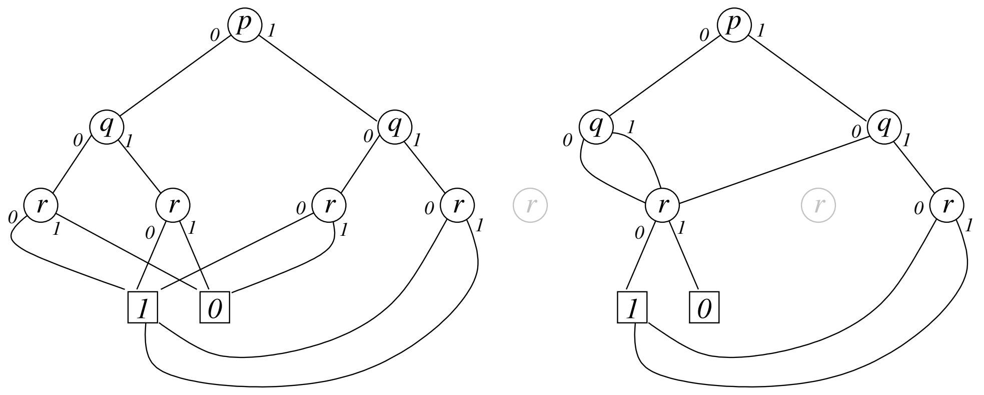
$$\overline{p}\,\overline{q}\,\overline{r} + \overline{p}\,q\,\overline{r} + p\,\overline{q}\,\overline{r} + p\,q\,\overline{r} + p\,q\,r$$

Another way to represent terms is to use a tree. In the tree below, the nodes are labeled with the names of variables occurring in E, and the edges with boolean values indicating whether the variable is true or false along that path.



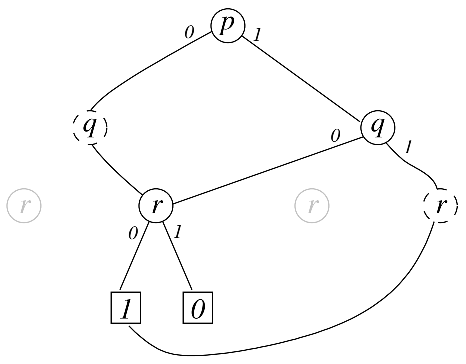
This is called a *binary decision tree* because each non-leaf has out-degree 2. We will see more general classes of decision trees in Chapter 4. Nodes are labeled with the variable whose value determines what path to follow; and edges are labeled with 1s and 0s.

A more compact representation of term E can be obtained by building a DAG to which the binary decision is homomorphic (Defn. 2.16. We can do this step-wise, from the bottom up, by locating isomorphic (Defn. 2.10) subtrees and eliminating all but one of them. This is done in two steps below, first eliminating redundant leaves, and then isomorphic subtrees rooted at r



So now we have a binary decision DAG in which every path from root to leaf corresponds to a path in the original tree. These paths, taken together, correspond to the clauses in a DNF.

In computer representations, additional transformations are used to make the representation still more compact. If the two branches from the node go to the same target, that node can be eliminated.



The resulting DAG is no longer homomorphic to the original tree (Can you see why?) unless the missing variable is somehow "remembered" when traversing the graph. The DAG above is called a *reduced ordered binary decision diagram* or *ROBDD* for short. Authors often abbreviate the acronym *ROBDD* to just *BDD*.

Suppose a boolean or propositional term contains n different variables. The size of its truth table is *always* proportional to 2^n. That is, the truth table becomes "exponentially large." The term's DNF *may* become exponentially large, depending on the number of true cases, but independent of the variable ordering. Likewise, the term's ROBDD *may* become exponentially large, depending which variable contribute to the term's value *and* on the variable ordering. All forms are used as computer representations, but ROBDDs are often the best choice.

Exercises 3.4

1. Construct truth tables and DNFs for the following propositional formulas

(a) $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$

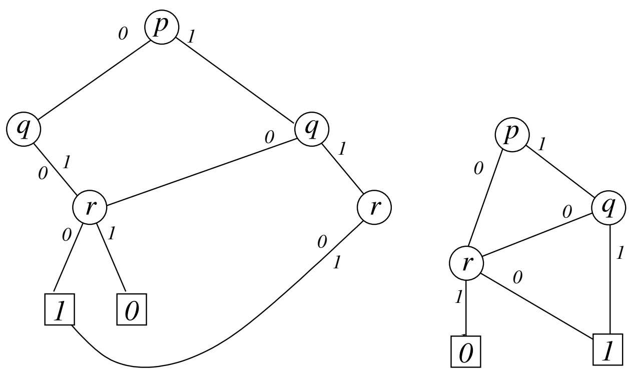
(b) $((P \Rightarrow R) \wedge (Q \Rightarrow R)) \Leftrightarrow ((P \wedge Q) \Rightarrow R)$

(c) $((P \Rightarrow R) \vee (Q \Rightarrow R)) \Rightarrow ((P \vee Q) \Rightarrow R)$

(d) $((P \Rightarrow R) \vee (Q \Rightarrow S)) \Rightarrow ((P \vee Q) \Rightarrow (R \vee S))$

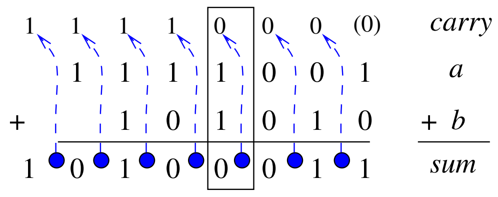
2. The term *disjunctive normal form* suggests that there might be such a thing as *conjunctive normal form* (*CNF*), and there is. What would the CNF of a formula look like? Devise a systematic way to synthesize a CNF from the truth table of a propositional formula.

3. In Section 3.4.1 it is claimed that the two graphs shown below are *not* homomorphic. Consult Definition 2.16 and explain why.



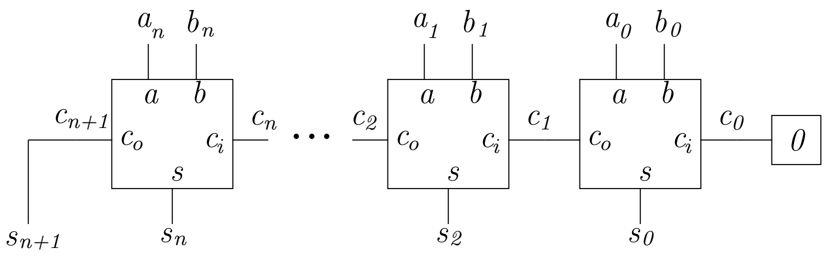
3.5 Application of Boolean Algebra to Hardware Synthesis*

The Boolean algebra for $\{1, 0\}$ has applications to the description of digital hardware. Addition of binary numbers involves the same column-by-column procedure as decimal addition, except that the arithmetic is base 2. For instance to add the numbers 1111001_2 and 101010_2,

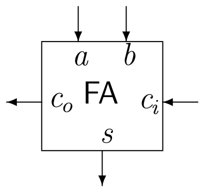


start at the least significant position adding the two right-most digits. If the result is 10_2 or 11_2, the "2's place" is carried into the next column. To make the algorithm uniform, we start out with a *carry-in* of 0; and if the most-significant *carry-out* is 1, another place is given to the sum.

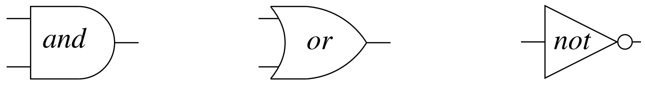
The procedure is implemented in hardware by connecting a series of identical single-bit *full adders*, one for each bit of the operands.



So building an n-bit adder requires designing a full adder and replicating it n times.



Digital hardware is built using a set of devices, or *logic gates*, that operate on two distinct voltage levels, called *high* (H) and *low* (L). The actual voltage values depend on the technology used to make the devices[1] The simplest of these devices realize the functions *and*, *or* and *not*, which are represented by the schematic symbols



The truth table below specifies what the Full Adder does. It has two outputs, s for the *sum* and c_o for the carry-out, so two truth tables are needed. required

a	b	c_i	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

If we express these tables in disjunctive normal form (sometimes called *sum-of-products* form in this context), a naive implementation is obtained, since we have logic gates to realize each operation.

$$\begin{array}{rcl} s & = & \overline{a}\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + a\,b\,c \\ c_o & = & \overline{a}\,b\,c + a\,\overline{b}\,c + a\,b\,\overline{c} + a\,b\,c \end{array}$$

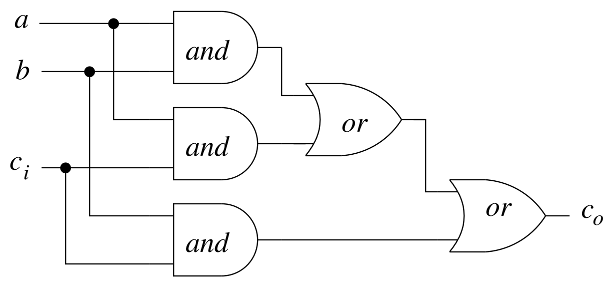
However, it would be better to reduce these formulas to something smaller, in order to use fewer gates. One way to do this is to reduce the expressions algebraically to equivalent but smaller formulas. The c_o output can be reduced

[1]In integrated circuits, H is 3–5 volts and L is 0 volts relative to a reference voltage called *ground*. Of course, in physical devices the voltages are not exact, and may range a bit from their ideal values.

to

$$
\begin{array}{rcll}
c_o & = & \bar{a}\,b\,c + a\,\bar{b}\,c + a\,b\,\bar{c} + a\,b\,c & \text{(truth table)} \\
& = & \bar{a}\,b\,c + a\,\bar{b}\,c + a\,b\,\bar{c} + (a\,b\,c + a\,b\,c) & \text{(idempotence)} \\
& = & \bar{a}\,b\,c + a\,b\,c + a\,\bar{b}\,c + a\,b\,\bar{c} + a\,b\,c & \text{(commutativity,} \\
& & & \text{associativity)} \\
& = & \bar{a}\,b\,c + a\,(\bar{b}\,c + b\,\bar{c} + b\,c) & \text{(distributivity)} \\
& = & \bar{a}\,b\,c + a\,b\,c + a\,(\bar{b}\,c + b\,c + b\,\bar{c} + b\,c) & \text{(idempotence)} \\
& = & \bar{a}\,b\,c + a\,b\,c + a\,((\bar{b} + b)\,c + b\,(\bar{c} + c)) & \text{(distributivity, twice)} \\
& = & \bar{a}\,b\,c + a\,b\,c + a\,(c + b) & \text{(idempotence)} \\
& = & (\bar{a} + a)\,b\,c + a\,(c + b) & \text{(distributivity)} \\
& = & 1\,b\,c + a\,(c + b) & \text{(cancellation)} \\
& = & b\,c + a\,(c + b) & \text{(identity)} \\
& = & b\,c + a\,c + a\,b & \text{(distributivity)}
\end{array}
$$

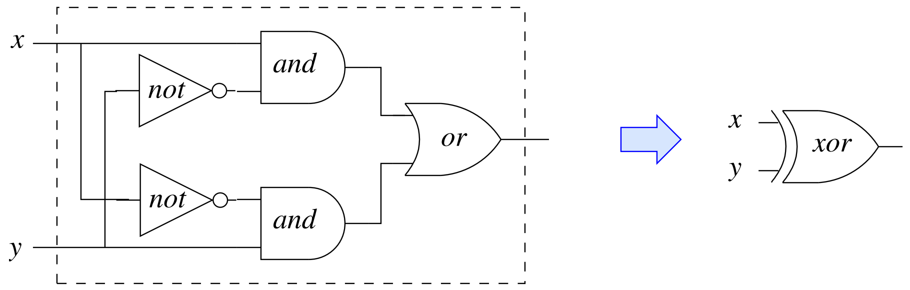
Hence, c_o is implemented with



The derivation of c_o followed the typical pattern of enlarging the formula so that it could later be simplified. Such algebraic manipulations are often done with a goal in mind, and reaching that goal may involve expansion, even if the ultimate objective is reduction.

"Simplification" of s is even more subtle. Suppose we have a device that realizes the the *exclusive-or* operation,

$$x \oplus y \overset{\text{def}}{=} \bar{x}\,y + x\,\bar{y}$$



The goal now is to derive an equivalent expression that uses instances of '$\oplus$'.

$$
\begin{aligned}
s &= \overline{a}\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + a\,b\,c && \text{(truth table)}\\
&= \overline{a}\,(\overline{b}\,c + b\,\overline{c}) + a\,(\,\overline{b}\,\overline{c} + b\,c\,) && \text{(distributivity, twice)}
\end{aligned}
$$

$$
\boxed{
\begin{array}{lll}
\overline{b}\,\overline{c} + b\,c & & \\
\quad = & \overline{\overline{\overline{b}\,\overline{c} + b\,c}} & \text{(negation)}\\
\quad = & \overline{\overline{(\overline{b}\,\overline{c})}\;\overline{(b\,c)}} & \text{(DeMorgan's Law)}\\
\quad = & \overline{(\overline{\overline{b}} + \overline{\overline{c}})\;(\overline{b} + \overline{c})} & \text{(DeMorgan's Law)}\\
\quad = & \overline{(b + c)\,(\overline{b} + \overline{c})} & \text{(negation)}\\
\quad = & \overline{(b + c)\overline{b} + (b + c)\,\overline{c})} & \text{(distributivity)}\\
\quad = & \overline{b\,\overline{b} + c\overline{b} + b\,\overline{c} + c\,\overline{c}} & \text{(distributivity)}\\
\quad = & \overline{0 + c\overline{b} + b\,\overline{c} + 0} & \text{(cancellation)}\\
\quad = & \overline{\overline{b}c + b\,\overline{c}} & \text{(identity,}\\
 & & \text{commutativity)}
\end{array}
}
$$

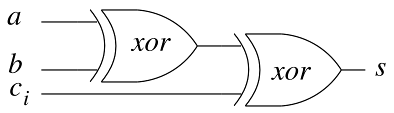
$$
\begin{aligned}
&= \overline{a}\,(\overline{b}\,c + b\,\overline{c}) + a\,\overline{(\overline{b}\,c + b\,\overline{c})} && \text{(boxed derivation)}\\
&\;\;\vdots
\end{aligned}
$$

A *subsidiary* derivation is used to refine the sub-formula $\overline{b}\,\overline{c} + b\,c$ to $\overline{b}\,c + b\,\overline{c}$.

Continuing with with the derivation,

$$\begin{array}{lll} & \vdots & \\ = & \overline{a}\,(\overline{b}\,c + b\,\overline{c}) + a\,\overline{(\overline{b}\,c + b\,\overline{c})} & \\ = & \overline{a}\,(b \oplus c) + a\,\overline{(b \oplus c)} & \text{(definition `}\oplus\text{')} \\ = & a \oplus (b \oplus c) & \text{(definition `}\oplus\text{' (!))} \end{array}$$

The implementation of s becomes



The second instance of '$\oplus$' involves sub-expressions rather than simple variables. So even in boolean algebra derivations can become very complex, as you already know from ordinary algebra.

It is possible but unlikely that the components of a real adder would be designed using algebraic derivations. Instead, the designer would "guess" an efficient implementation of the truth tables and *verify* it against the original specification. The verification process involves comparison of normal forms such as ROBDDs.

Exercises 3.5

1. Draw two distinct ROBDDs for the output s_1 of the 2-bit adder, given by

$$\begin{array}{lll} s_1 & = & d \oplus e \oplus c_1 \\ & = & d \oplus e \oplus (a\,b + a\,c + b\,c) \end{array}$$

2. Draw the ROBDD for output s_2 of the 2-bit adder under the variable order $\langle c, b, a, e, d\rangle$. [HINT. *Work top-down, simplifying as you go along. Suppose that a node labelled* v *represents a term* Φ. $\Phi = \overline{v} \cdot \Phi_0 + v \cdot \Phi_1$, *in which* Φ_0 *and* Φ_1 *are obtained by evaluating* Φ *with* $v = 0$ *and* $v = 1$, *repectively.*]

3. The derivation of sum bit s yields

$$\overline{a}\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + a\,b\,c \;=\; a \oplus (b \oplus c)$$

Expanding the definition of '$\oplus$' yields

$$\overline{a}\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + a\,b\,c \;=\; \overline{a}(b\,\overline{c} + \overline{b}\,c) \oplus a(\overline{b\,\overline{c} + \overline{b}\,c})$$

Choose a variable ordering, such as $\langle a,\, b,\, c\rangle$. Construct and compare ROBDDs for both sides of the equation above. .

Chapter 4

Counting

One often wants to know how many elements a set contains. In fact, knowing this number is often more important than knowing just what the elements are! Definition 4.1, introduces notation and terminology relating to a set's size.

Section 4.3 lays a foundation for *counting* the elements in a set. This is usually done by posing a kind of experiment in which the question, "How many elements are in set S?" is decomposed into a sequence of simpler decisions for which the number of outcomes is already known. One then tallies all the outcomes to get a final answer.

4.1 Extended Operations

Suppose we are given a set of numbers, $S = \{n_1, n_2, \ldots, n_k\}$, and wish to express their sum. One way to do it is to use ellipses:

$$n_1 + n_2 + \cdots + n_k$$

We might use a program:

```
$\mathcal{P}$: begin
    $i$ := $0$;
    $z$ := $0$;
$\ell_1$: while $i < k$ do
      begin
      $i$ := $i + 1$;
      $z$ := $z + n_i$
      end
    {$z = n_1 + \cdots + n_k$}
```

You have probably also seen *summation notation*:

$$\sum_{i=1}^{k} n_i$$

The variable i in the formula above is called an *index variable* and is understood to range over integers between the *lower bound* 1 and the *upper bound* k. That is, $1 \leq i \leq k$.

The set S could have been defined as

$$S = \{n_i \mid 1 \leq i \leq k\} = \{n_1, n_2, \ldots, n_k\}$$

And likewise, the summation might be expressed in several other ways, such as

$$\sum \{n_1, n_2, \ldots, n_k\}$$

or

$$\sum \{n_i \mid 1 \leq i \leq k\}$$

or even simply

$$\sum S.$$

In other words, we may express summation over *any* set of numbers, indexed or not. So what about $\sum \emptyset$? By convention (because it works out) we define summation over the empty set as

$$\sum \emptyset = 0$$

For example, a sum of the first 101 reciprocal powers of two could be written as

$$\sum_{i=0}^{100} \frac{1}{2^i} \quad \text{or} \quad \sum \{2^{-i} \mid 0 \leq i \leq 100\} \quad \text{or even} \quad \sum_{0 \leq i \leq 100} \frac{1}{2^i}$$

In the rightmost formula, the summation bounds have now been incorporated in an inequality expression, but they mean the same thing. The infinite sum $\sum \{2^{-i} \mid i \in \mathbb{N}\}$ could be written as

$$\sum_{i=0}^{\infty} \frac{1}{2^i} \quad \text{or as} \quad \sum_{i \in \mathbb{N}} \frac{1}{2^i}$$

Sometimes, indices range over sets that are not simple intervals of the number line. In fact, the indices need not be numbers at all. See Example 4.2.

Other operations can be extended in a manner similar to addition and multiplication. The requirements are that the operation be a commutative, associative function with an identity (Defn. 2.7) element for cases where the index set is empty.

Example 4.1 Some of the extended opertions arising in this book are shown in the following examples. In each example, I stands for some index set, and $S = \{x_i \mid i \in I\}$ is an indexed set. Note that saying "I is empty," is equivalent to saying "S is empty."

(a) Let $S = \{n_i \mid i \in I\}$ be an indexed set of numbers. The *extended product*

$$\prod S \text{ is the multiplicative product of all the numbers in } S$$

For the empty case, $\prod \emptyset = 1$.

(b) Let $S = \{X_i \mid i \in I\}$ be an indexed collection of sets. The *extended union*

$$\bigcup_{i \in I} X_i = \{x \mid x \in A_j \text{ for some } j \in I\}$$

For the empty case, $\bigcup \emptyset = \emptyset$.

(c) Let U be a predetermined set. let $S = \{A_i \mid i \in I\}$ be an indexed collection of *subsets* of U. The *extended intersection*

$$\bigcap S = \{a \mid a \in X_j \text{ for all } j \in I\}$$

For the empty case, $\bigcap \emptyset = U$.

(d) Let $S = \{P_i \mid i \in I\}$ be an indexed collection of propositions. The *extended conjunction* of these propositions is

$$\bigwedge S = \begin{cases} T & \text{if } P_i \text{ is } T \text{ for all } i \in I \\ F & \text{if for some } j \in I \ P_j \text{ is } F \end{cases}$$

with $\bigwedge \emptyset = U$.

(e) Let $S = \{P_i \mid i \in I\}$ be an indexed collection of propositions. The *extended disjuncton of* S is

$$\bigvee S = \begin{cases} T & \text{if for some } j \in I \ P_j \text{ is } T \\ F & \text{if } P_j \text{ is } F \text{ for all } j \in I \end{cases}$$

with $\bigvee \emptyset = F$.

□

4.2 Cardinality

Definition 4.1 *$|S|$ denotes the number of elements in S. $|S|$ is called the* size *or* cardinality *of S.*

For example, $|\{a,b,c,d,e\}| = 5$, $|\emptyset| = 0$; and

$$|\{p \mid p \text{ is a prime number less than } 30\}| = 10$$

Fact 4.1 *Let A and B be finite sets.*

(a) $|A \cup B| = |A| + |B| - |A \cap B|$

(b) $|A \cap B| = |A| + |B| - |A \cup B|$

(c) $|A \times B| = |A| \cdot |B|$

(d) $|A \setminus B| = |A| - |A \cap B|$

Comparing this list with the set operations defined in Definition 1.5, $|\mathcal{P}(A)|$ is missing. We will consider $|\mathcal{P}(A)|$ later.

Example 4.2 The formula

$$\sum_{X \in \mathcal{P}(A)} |X|, \text{ or equivalently, } \sum_{X \subseteq A} |X|$$

expresses the sum *of the sizes of all subsets* of A. For instance, if $A = \{a, b, c\}$, then

$$\begin{aligned}
&\sum_{X \in \mathcal{P}(A)} |X| \\
&\quad = |\emptyset| + |\{a\}| + |\{b\}| + |\{c\}| + |\{a,b\}| + |\{a,c\}| + |\{b,c\}| + |\{a,b,c\}| \\
&\quad = 0 + 1 + 1 + 1 + 2 + 2 + 2 + 3 \\
&\quad = 12
\end{aligned}$$

□

4.3 Permutations and Combinations

Let $A = \{a, b, c\}$ an alphabet. How many three-letter words can be made from letters in A? One way to look at the problem is to draw three boxes and consider how many choices there are to fill each box with a letter:

$$\square\square\square$$

There are three choices, a or b or c, for each box, so the number of three letter words is $3 \times 3 \times 3 = 27$.

aaa	*aab*	*aac*	*aba*	*abb*	*abc*	*aca*	*acb*	*acc*
baa	*bab*	*bac*	*bba*	*bbb*	*bbc*	*bca*	*bcb*	*bcc*
caa	*cab*	*cac*	*cba*	*cbb*	*cbc*	*cca*	*ccb*	*ccc*

How many three-letter words are there in which no letter is repeated? Try it yourself, and compare your answer with

abc
acb

bac
bca

cab
cba

The difference between these two examples is that, in the first, the choice of any letter is *independent* of the

other choices; and in the second, the choice of each letter *depends* on the other choices. Let $B = \{a, b, c, d\}$ and consider the different four-letter words from B without repeated letters. Try it yourself, and compare your answer with

$abcd$	$bacd$	$cabd$	$dabc$
$abdc$	$badc$	$cadb$	$dacb$
$acbd$	$bcad$	$cbad$	$dbac$
$acdb$	$bcda$	$cbda$	$dbca$
$adbc$	$bdac$	$cdab$	$dcab$
$adcb$	$bdca$	$cdba$	$dcba$

The listing is organized according to the choice of which letter is first. Having made that choice, all orderings for the remaining three letters are listed. This is a problem we have already solved: there are six possibilities. Figure 4.1(a) shows a tree, labeled to show how the listing is organized. Each path *through* the tree (i.e., from the root to a leaf) represents one letter ordering, as determined by reading the edge labels in order along the path from the root to a leaf. Trees used in this way are called *decision trees*.

If we are interested only in the number of solutions, and not what they are, the decision tree can be reduced in size by taking symmetries into account. In Figure 4.1(a) the subtrees at any level are isomorphic (Defn. 2.10, differing only in their labels. So instead of drawing all of them, we just keep track of how many there are. Figure 4.1(b) depicts a decision tree in which the edges are labeled with the number of isomorphic subtrees they represent. The product of the numbers along a path gives the number of solutions represented.

How many letter-orderings does a five-letter alphabet have? There are five choices for the first letter, and we have already shown that there are 24 ways to order the remaining four letters. Hence, the answer to the five-letter question is $5 \times 24 = 120$.

Let us generalize this discussion.

Fact 4.2 *Let S be a set of size n. There are*

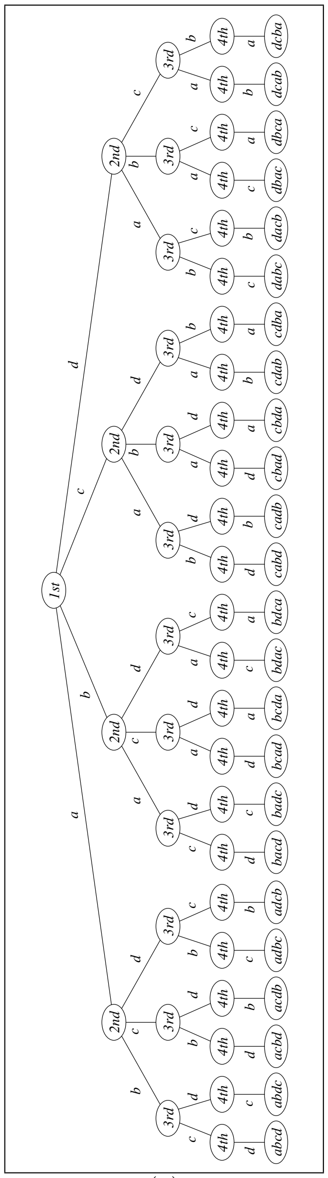
$$1 \times 2 \times \cdots \times n$$

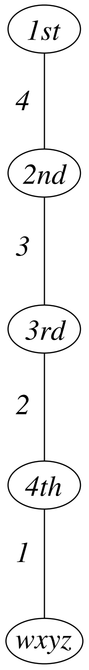
different orders in which the elements of S can be listed. Such a listing, in which each $s \in S$ occurs exactly once, is called permutation of *S.*

The "product of whole numbers from 1 to n" is useful enough to be given its own notation:

Definition 4.2 *The product*

$$\prod_{k=1}^{n} k = 1 \times 2 \times \cdots \times n$$


(a)



(b)

Figure 4.1: A decision tree (a) and counting tree (b) for ordering a 4-letter alphabet

is called n factorial*, written $n!$. By convention (Recall Example 4.1(a)),* $0!$ *is defined to be 1.*

Thus, if $|S| = n$ it has $n!$ permutations.

How many ways are there to list two distinct elements from $A = \{a, b, c, d\}$? Without actually doing it, we could reason as before that

1. There are four possibilities for the first letter.
2. Once the first letter is chosen, three possibilities remain for the second.

So the number of two-letter orderings is $4 \times 3 = 12$. Counting by taking the product of the successive outcomes is called the *Rule of Products* in some textbooks, and the *Principle of Choice* in others.

Another way to look at the problem problem is to start with something you already know—the number of permutations of a set of four elements—and eliminate rudundant representatives. The listing below strikes out all but one of the permuations whose first two letters are the same.

abcd	*bacd*	*cabd*	*dabc*
~~*abdc*~~	~~*badc*~~	~~*cadb*~~	~~*dacb*~~
acbd	*bcad*	*cbad*	*dbac*
~~*acdb*~~	~~*bcda*~~	~~*cbda*~~	~~*dbca*~~
adbc	*bdac*	*cdab*	*dcab*
~~*adcb*~~	~~*bdca*~~	~~*cdba*~~	~~*dcba*~~

So the number of 2-letter permutations is equal to the number of 4-letter permutations divided by the number which represent the same 2-letter outcome. Again, we are taking advantage of a symmetry in the problem, knowing that the partitioning is uniformly independent of what the first two letters actually are. Definition 4.3, below, summarizes this discussion.

Definition 4.3 *Let A be a set with $|A| = n$. For $m \leq n$, the number of distinct ways to list m distinct elements from A is*

$$\frac{n!}{(n-m)!}$$

*Such a listing is called an m-*permutation*.*

Example 4.3 *You have 26 trophies you would like to display across your fireplace mantel, but there is for only room 10 of them. How many different ways can you do this?*

There are 10 positions at which a trophy can be placed, so the question is simply asking how many 10-permutations are there for a set with 26 elements. By Definition 4.3 this number is

$$\frac{26!}{16!}$$

or $26 \times 25 \times \cdots \times 17$. Unless you have a computer handy and the time, don't bother to calculate this number; it is 19,275,223,968,000.

$\square$

Suppose you've placed 10 trophies on the mantel, and you decide it is better to place the largest one in the middle. Does swapping two of the selected trophies result in a "different" display? The problem statement fails to make this clear, but the solution given presumes that the order of the trophies matters.

Now suppose we are interested in selecting two letters from $A = \{a, b, c, d\}$ but don't care about the order. In other words we are asking how many distinct *subsets of size 2* are there in a set of four elements. As before, we can consider the set of A's 2-permutations and strike out the redundant ones—those having the same two first letters in either order.

abcd	~~*bacd*~~	~~*cabd*~~	~~*dabc*~~
~~*abdc*~~	~~*badc*~~	~~*cadb*~~	~~*dacb*~~
acbd	*bcad*	~~*cbad*~~	~~*dbac*~~
~~*acdb*~~	~~*bcda*~~	~~*cbda*~~	~~*dbca*~~
adbc	*bdac*	*cdab*	~~*dcab*~~
~~*adcb*~~	~~*bdca*~~	~~*cdba*~~	~~*dcba*~~

Definition 4.4 *The* chose *function,* $\binom{n}{k}$ *is the number of different ways to choose a subset of k elements from a set of size n. This number is given by the formula*

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$$

$\binom{n}{k}$ is calculated by dividing the number of permutations of an n-element set by $k!$, the number of ways to permute the first k elements times $(n-k)!$ the number of ways to permute the remaining elements beyond the k^{th}. $\binom{n}{k}$ is also known as the *binomial coefficient* or the *combinational number*.

Example 4.4 *You have 26 trophies you would like to display across your fireplace mantel, but there is for only room 10 of them. How many different ways can you do this?*

This is the same problem as Ex. 4.3, which took the ordering of the ten trophies into account. If the order doesn't matter, we should divide out the number of ways they can be arranged on the mantel:

$$\frac{19,275,223,968,000}{10!} = \frac{19,275,223,968,000}{3,628,800} = 5,311,735 = \frac{26!}{10! \times 16!} = \binom{26}{10}$$

$\square$

Example 4.5 *In how many ways can a set six elements, $A = a, b, c, d, ef$, be partitioned into three subsets, X, Y and Z, containing three, two and and elements, respectively?*

Solve this problem by counting the partitions one at a time:

(a) By Definiton 4.4 there are

$$\binom{6}{3} = \frac{6!}{3! \times 3!} = \frac{6 \cdot 5 \cdot 4}{3 \cdot 2 \cdot 1} = 20$$

ways to choose three elements from A.

(b) Once X has been determined, there are three elements left to choose for Y. The number of ways to do that is

$$\binom{3}{2} = \frac{3!}{2! \times 1!} = 3$$

(c) Once X and Y have been determined, there is one remaining element to choze for Z

$$\binom{1}{1} = \frac{1!}{1! \times 0!} = \frac{1}{1 \cdot 1} = 1$$

(d) The number of partitionings is the product of the numbers of these choices, $20 \cdot 3 \cdot 1 = 60$.

It shouldn't matter what order you choose X, Y and Z. Check that

$$\binom{6}{1} \times \binom{5}{2} \times \binom{3}{3} = \binom{6}{2} \times \binom{4}{1} \times \binom{3}{3} = \; etc.$$

□

In Step (b) of Ex. 4.5 calculated that there were $\binom{3}{2}$ ways to choose two elements from 3 for Y. An equivalent problem is to choose one element *not* to include in Y, and there are $\binom{3}{1}$ ways to do that. In general,

Proposition 4.3

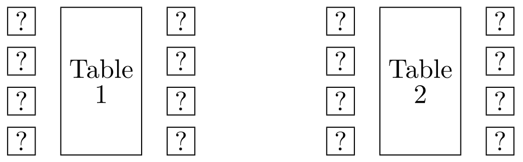
$$\binom{n}{k} = \binom{n}{n-k}$$

PROOF: By Definition 4.4 and since $n - (n - k) = k$,

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!} = \frac{n!}{(n-k)! \times k!} = \frac{n!}{(n-k)! \times (n-(n-k))!} = \binom{n}{n-k}$$

□

Example 4.6 You're having a dinner party for sixteen guests, including yourself, of which half are male and half are female. You have two tables each seating eight, with four places on two sides:



You want to make sure that both tables have an equal number of guys and gals. Bob is coming, but you don't know about Jane yet. If Jane does come you need to make sure that she and Bob sit at different tables. How many ways are there to do this?

Solution: There are two cases to consider, according to whether or not Jane comes to the party

1. If Jane is not coming, then anyone can sit at either table. You need to choose four guys and four gals to sit at Table 1. The number of ways to do this is
$$\binom{8}{4} \times \binom{8}{4}$$
Once the people sitting at Table 1 are determined, all the rest will be assigned to Table 2, and there is only one way to do that.

2. If Jane comes she must sit at one table and Bob at the other.

   (a) Suppose Jane is assigned to Table 1. Then you need to choose:

      i. 3 other gals from the remaining 7 to sit at Table 1. There are $\binom{7}{3}$ ways to do that.

      ii. Four guys from the remaining set of 7—Bob is excluded—will sit at Table 1. There are $\binom{7}{4}$ ways to do that.

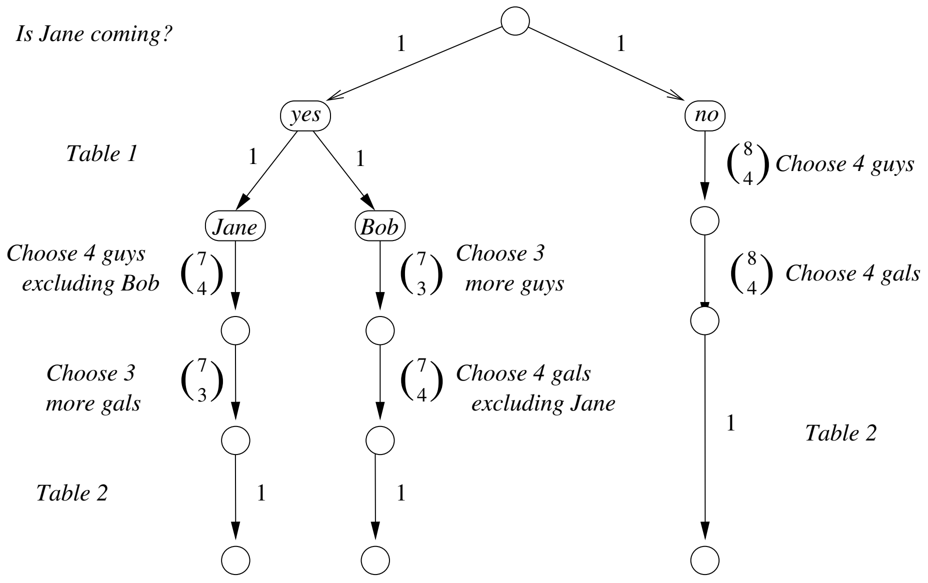
      iii. Once the assignments are made to Table 1, all the remaining guests sit at Table 2, so there is only just one choice.

   So there are $\binom{7}{3} \times \binom{7}{4}$ seating assignments in this case.

   (b) If Jane is assigned to Table 2, it's the same problem. So the number of assignments in this case is also $\binom{7}{3} \times \binom{7}{4}$.

Since $\binom{7}{3} = \binom{7}{4}$ the answer is that there are
$$\binom{8}{4} \times \binom{8}{4} + \binom{7}{4} \times \binom{7}{4} + \binom{7}{4} \times \binom{7}{4}$$
ways to assign people to tables. □

You may already be thinking that there are many more ways to assign *seating* for this party. Once 8 people are selected for a table, there are 8! ways to arrange them. In the next example we will consider seating assignments. A counting tree for Example 4.6 might look like


Is Jane coming?
1
1
yes
no
Table 1
1
1
$\binom{8}{4}$ Choose 4 guys
Jane
Bob
Choose 4 guys excluding Bob $\binom{7}{4}$
$\binom{7}{3}$ Choose 3 more guys
$\binom{8}{4}$ Choose 4 gals
Choose 3 more gals $\binom{7}{3}$
$\binom{7}{4}$ Choose 4 gals excluding Jane
1
Table 2
Table 2
1
1

The total number of solutions for a problem is the sum, over all the paths *through* (i.e., from the root to a leaf) its counting tree of the product of numbers along each path. More formally,

Fact 4.4 *let $T \subseteq A \times A$ be a counting tree with root r and edge labeling $L\colon T \to \mathbb{N}$. Let $P = \{\langle r, a_1, \ldots, \ell\rangle \mid \ell$ a leaf in $T\}$. The number of solutions represented by T is*

$$\sum_{p\in P}\Big(\prod_{e\in p} L(e)\Big)$$

The formula above takes some liberties with notation. The indexing specifier "$e \in p$" is saying "take all the *edges* in (or along) *path p*." Although in Chapter 2, paths are defined to be sequences, not sets, the meaning is should be clear.

Example 4.7 For the same party as Example 4.6, how many ways are there to assign guests to seats in such a way that every guy is sitting next to at least one gal, and *vice versa*.

SOLUTION:

(1) *Assign guests to tables as in Example 4.6*

(2) *Pick a table*

(3) *Pick gender arrangement for one side from* {MFMF, MFFM, FMFM, FMMF}

(4) *Pick gender arrangement for the other side from* {MFMF, MFFM, FMFM, FMMF}

(5) *Assign 4 guys to 4 seats*

(6) *Assign 4 gals to 4 seats*

$$\begin{tikzcd}
\textcircled{1} \arrow[d, "\binom{8}{4}^2 + 2\binom{7}{4}^2"] \\
\textcircled{2} \arrow[d, "2"] \\
\textcircled{3} \arrow[d, "4"] \\
\textcircled{4} \arrow[d, "4"] \\
\textcircled{5} \arrow[d, "4!"] \\
\textcircled{6} \arrow[d, "4!"] \\
\bullet
\end{tikzcd}$$

Thus, for each assignment of guests to tables, there are $2 \cdot 4 \cdot 4 \cdot 4! \cdot 4! = 18,432$ different ways to assign guest to seats in such a way that each guy is sitting next to at least one gal and *vice versa*. The final count becomes

$$18432 \cdot \left[\binom{8}{4}^2 + 2\binom{7}{4}^2 \right]$$

□

Exercises 4.3

1. Suppose you want to assign seats for a single row of 4 guys and 4 gals in such a way that each guy is sitting next to *at least* one gal, and *vice versa*. How many ways are there to do this? HINT: Use a decision tree, and practice by solving the 3-guy, 3-gal problem.

2. In Example 4.7, suppose you want to assign seats so that each guy is sitting next to *or across from* at least one gal, and *vice versa*. How many ways are there to do this? HINT: List out all the gender arrangements.

3. A standard deck of cards has 52 cards consisting of 13 cards in each of four *suits*: ♠, ♡, ♢ ♣. In each suit, cards have *face value*s from $\{1, 2, \ldots, 13\}$, each card having a different face value. A *hand* is a set of five cards from the deck. A hand is called a *flush* if all five cards are of the same suit. A hand is called a *straight* if the five cards are sequential in value, for instance, $\{3\heartsuit, 4\spadesuit, 5\diamondsuit, 6\diamondsuit, 7\heartsuit\}$.

(a) How many different flushes are there in a standard deck?

(b) How many different straights are there in a standard deck?

4. *Prove:* For all $n, k \in \mathbb{N}$, $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$

5. If $|A| = n$ guess the value of $|\mathcal{P}(A)|$ by listing a few small examples.

4.4 Probability, Briefly

A variation of counting trees can be used to determine the liklihood of outcomes for certain kinds of "experiments."

Definition 4.5 *In an experiment where individual outcomes are equally likely at any step, the* probability *of a given outcome is*

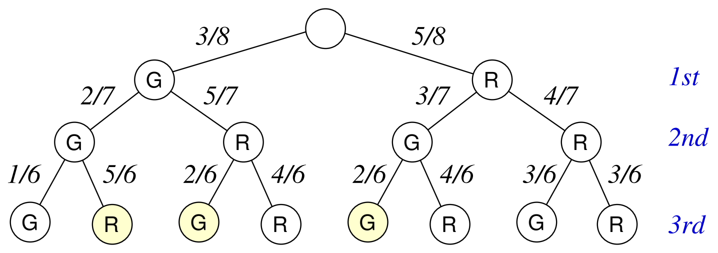
$$\Pr(E) = \frac{n}{t}$$

where n is the number of ways a given outcome E can occur and t is the total number of outcomes.

A *probability tree* is (like) a counting tree whose branches are labeled with probabilities rather than counts. The *path probability* in a probability tree is the product of probabilities along that path. The probability of a result is the sum of the path probabilities leading to that result.

Example 4.8 *You have a bag containing five red marbles and three green marbles. From this bag you blindly select three marbles, one at a time. What is the probability that just two of them are green?*

The probability tree below represents the "experiment" of picking three marbles from the bag. At the beginning, there are eight marbles in the bag and three of them are green. It is equally likely that any marble will be picked, so $\Pr(G) = 3/8$. Once the first marble has been choosen, the probabilities change. Suppose the first marble is green. Now there are two green marbles among the seven left in the bag, so $\Pr(G) = 2/7$.



$\square$

Chapter 5

Numerical Induction

The mathematics of programming—indeed, programming itself—is deeply related to inductive reasoning. It is the most important topic in this book. The word "induction" derives from the notion of learning from seeing, or arriving at a generalization from observation of instances—a fundamental aspect of science, certainly, and perhaps human experience as well. The notion of numerical induction involves *proving* some numbers have a property (*base cases*) and then showing *how to prove* the rest do (the *induction*).

Various styles of inductive reasoning arise in computer science. As we shall see in Chapter 7, induction may be applied to sets other than $\mathbb{W}$, but the arguments can be reduced to the numerical induction principle introduced in this chapter. One may look at it in another way, though, and see that certain kinds of sets come with a "built-in" induction principle, and $\mathbb{W}$ is one such set.

Numerical induction is a way of proving *"For all $n \in \mathbb{W}$, $H[n]$"* for some property H. In words, the induction principle says,

NUMERICAL INDUCTION

If you can prove

(BASE CASE) $H[1]$ holds.

and

(INDUCTION STEP) For an arbitrary $k \in \mathbb{W}$, $H[k]$ implies $H[k+1]$.

then you may conclude by induction that

$H[n]$ holds for all $n \in \mathbb{W}$.

The BASE CASE is usually proved by a direct argument about 1. In the INDUCTION STEP, $H[k]$ *implies* $H[k+1]$, the premise $H[k]$ is called the *induction hypothesis*. The argument for the induction case must in no way depend on the choice of k We are actually proving: *"For all* $k \in \mathbb{W}$*,* $H[k] \Rightarrow H[k+1]$. In particular, the induction step must be valid for $k = 1$.

5.1 First Examples

Example 5.1 Prove: *For all* $n \in \mathbb{W}$, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

PROOF. The PROOF is by induction on $k \in \mathbb{W}$ with hypothesis

$$H[k] \equiv \sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

BASE CASE. For $H[1]$

$$\sum_{i=1}^{1} i = 1 = \frac{1 \cdot 2}{2}$$

Both sides of the equation simplify to 1, proving the base case.

INDUCTION STEP. Assume

$$H[k] \equiv \sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

We want to show it follows that

$$H[k+1] \equiv \sum_{i=1}^{k+1} i = \frac{(k+1)((k+1)+1)}{2} = \frac{(k+1)(k+2)}{2}$$

Starting from the left-hand side, we have

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) && \text{(expand } \Sigma \text{ once)} \\
&= \frac{k(k+1)}{2} + k + 1 && \textit{(induction hypothesis, } H[k]\textit{)} \\
&= \frac{k(k+1)}{2} + \frac{2k+2}{2} && \text{(multiply } k+1 \text{ by } \tfrac{2}{2}\text{)} \\
&= \frac{k^2+3k+2}{2} && \text{(add fractions, simplify)} \\
&= \frac{(k+1)(k+2)}{2} && \text{(factor the numerator)} \\
&= \frac{(k+1)((k+1)+1)}{2} && \text{(arithmetic)}
\end{aligned}$$

as needed. This proves the induction step, and we may now conclude, *"For all $n \in \mathbb{W}$, $\sum_{i=1}^{n} i = n(n+1)/2$," as desired* □

Remark. An intuitive form of this proof is attributed to Eighteenth Century mathematician Karl Friedrich Gauss. The story goes that as a punishment, the teacher assigned Gauss's 3rd grade class to add the numbers from 1 to 100. To the teacher's great surprise, Gauss turned in his answer after just a few minutes. He explained that writing the sum from left to right and and then again from right to left:

$$\begin{array}{rrrrrrrrr}
 & \Sigma i = & 1 & + & 2 & + & \cdots & + & 100 \\
+ & \Sigma i = & 100 & + & 99 & + & \cdots & + & 1 \\
\hline
= & 2\Sigma i = & 101 & + & 101 & + & \cdots & + & 101
\end{array}$$

Gauss saw that each column has the same partial sum so the final sum is $(100 \times 101) \div 2 = 5050$. The intuitive explanation is at least as compelling as the induction proof, wouldn't you agree?

□

The outline of an inductive proof is shown in Figure 5.1. You should include this "boilerplate" when doing the exercises in this section, even though some of it is repetitive. It will help clarify some of the variations on inductive arguments we explore later on.

THEOREM For all $n \in \mathbb{W}$, $H[n]$.

> *[$H[n]$ states a property over whole numbers. Later we will develop induction principles for other kinds of sets.]*

PROOF The proof is by induction on $k \in \mathbb{W}$ with hypothesis $H[k]$.

> *[Always announce the induction; it helps orient the reader. The induction hypothesis is the unquantified property $H[k]$. Any variable could be used in place of k—even n—but it is less confusing to introduce a new variable. Always state the variable over which the induction is done and state exactly what $H[k]$ is.]*

BASE CASE

> *[Argue directly for the truth of $H[1]$.]*

This concludes the base case.

INDUCTION STEP Assume $H[k]$.

> *[State the induction hypothesis. The proof of $H[k+1]$ may use this assumption once or several times. Always say exactly where the assumption is used.]*

This concludes the induction. We may conclude that $H[n]$ holds for all $n \in \mathbb{W}$.

Figure 5.1: Outline of an induction proof

Example 5.2 Proposition. *For given constants, a and $r \neq 1$, and for all whole numbers n,*

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

PROOF. The proof is by induction on k with hypothesis

$$H[k] \equiv \sum_{i=0}^{k} ar^i = \frac{ar^{k+1} - a}{r - 1}$$

BASE CASE. We are to prove that $H[1] \equiv ar^0 + ar^1 = (ar^{1+1} - a)/r - 1$. Reasoning from the right-hand side,

$$\begin{aligned}
\frac{ar^{(1+1)} - a}{r - 1} &= \frac{ar^2 - a}{r - 1} && (r^{(1+1)} = r^2) \\
&= \frac{a(r^2 - 1)}{r - 1} && (\text{distribute } a) \\
&= \frac{a(r + 1)(r - 1)}{r - 1} && (\text{factor } r^2 - 1) \\
&= a(r + 1) && (\text{cancel } r - 1) \\
&= ar + a && (\text{distribute } a) \\
&= ar^1 + ar^0 && (\text{substitute, } r^0 = 1,\ r^1 = r)
\end{aligned}$$

This proves the base case.

INDUCTION STEP. Assume $H[k] \equiv \sum_{i=0}^{k} ar^i = (ar^{k+1} - a)/(r - 1)$. To prove $H[k + 1]$,

$$\begin{aligned}
\sum_{i=0}^{k+1} ar^i &= \left(\sum_{i=0}^{k} ar^i\right) + ar^{k+1} && (\text{expand } \Sigma) \\
&= \frac{ar^{k+1} - a}{r - 1} + ar^{k+1} && (\textit{induction hypothesis, } H[k]) \\
&= \frac{ar^{k+1} - a}{r - 1} + \frac{ar^{k+1}(r - 1)}{r - 1} && (\text{multiply by } 1 = \frac{(r - 1)}{(r - 1)}) \\
&= \frac{ar^{k+1} - a + ar^{k+1}(r - 1)}{r - 1} && (\text{add fractions}) \\
&= \frac{ar^{k+1} - a + ar^{k+1}r - ar^{k+1}}{r - 1} && (\text{multiplication}) \\
&= \frac{ar^{k+1}r - a}{r - 1} && (\text{cancel negatives}) \\
&= \frac{ar^{k+2} - a}{r - 1} && (\text{add exponents})
\end{aligned}$$

This completes the induction case. We have shown by induction that

$$\textit{For all whole numbers } n, \ \sum_{i=0}^{n} ar^i = (ar^{n+1} - a)/(r-1)$$

□

REMARK. Both the base and induction cases show more details of algebra than would usually be required. From now on, algebraic derivations will be shorter. However, one must *always* show precisely where the induction hypothesis is applied.

□

Example 5.3 Assume that each number M is can be written as a product of prime numbers, $M = q_1 \cdot q_2 \ \cdots \ q_k$ and that this decomposition is unique. Prove:

Proposition. *There are infinitely many prime numbers.*

PROOF. The proof is by induction on k with hypothesis

$$H[k] \equiv \textit{“There are more than } k \textit{ prime numbers.”}$$

BASE CASE. $H[1]$ holds because there are more than 1 prime numbers. In particular, 2 and 3 are prime. This concludes the base case.

INDUCTION STEP. Assume that there are more than k prime numbers ($H[k]$). To prove that there are more than $k+1$ prime numbers ($H[k+1]$), let us assume there are *not* and derive a contradiction.

If there are more than k but not more than $k+1$ prime numbers, then there must be exactly $k+1$ primes. Let these numbers be p_1, p_2, ..., p_{k+1}. Now consider the number

$$m = 1 + p_1 \cdot p_2 \cdot \ \cdots \ \cdot p_{k+1}$$

m is not equal to any of the $k+1$ primes so it cannot be prime. Therefore, m must be evenly divisible by at least one prime number, p_i. But if p_i evenly divides n, and since it divides $1 + p_1 \cdot p_2 \cdot \ \cdots \ \cdot p_{k+1}$ it must also divide 1. This is impossible and so we have contradicted the assumption that there are just $k+1$ primes. There must be more, and that is just what we wanted to prove to complete the induction step and conclude that

$$\textit{For all } n \in \mathbb{W}, \textit{ there are more than } n \textit{ primes.}$$

Another way to state this conclusion is, *“There are infinitely many primes.”*

□

Exercises 5.1

1. The examples in this section are numerical inductions over $\mathbb{W} = \{1, 2, 3, \ldots\}$. With little or no modification these proofs can be made into numerical inductions over $\mathbb{N} = \{0, 1, 2, \ldots\}$ in which the base case is $H[0]$. For instance, Example 5.1 could be converted to *For all* $n \in \mathbb{N}$, $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$. Notice that lower range in the summation has been changed from $i = 1$ to $i = 0$. The BASE CASE becomes $\sum_{i=0}^{0} i = 0 = \frac{0\cdot(0+1)}{2}$. One should also verify that the proof of the INDUCTION STEP is valid for $k = 0$, that is, when applied to $H[0]$ implies $H[1]$

$$\sum_{i=0}^{0+1} i = \left(\sum_{i=0}^{0} i\right) + 0 + 1 \overset{\text{IH}}{=} \frac{0(0+1)}{2} + 0 + 1 = \cdots = \frac{(0+1)((0+1)+1)}{2}$$

Modify and check Examples 5.2 and 5.3 to obtain valid inductions over $\mathbb{N}$ rather than $\mathbb{W}$..

5.2 Base Translation

Sometimes, inductions start at some number other than 1. A proposition,

$$\text{“}\textit{For all } n \geq M \in \mathbb{W},\ H[n]\text{”}$$

could be rewritten, *“For all* $n \in \mathbb{W}$*, if* $n \geq M$ *then* $H[n]$*”* and proven, as before by numerical induction on n, with hypothesis

$$H'[k] \equiv \text{“}\textit{If } k \geq M \textit{ then } H[k]\text{”}$$

The base case reduces logically to proving $H[1]$.

Example 5.4 Proposition. *Any amount of postage exceeding* $7¢$ *can be made up using only* $3¢$ *and* $5¢$ *stamps.*

For example, a postage of $29¢$ is made up of eight $3¢$ stamps and one $5¢$ stamp.

PROOF. The proof is by induction on k with hypothesis

$H[k] \equiv$ *Any postage of up to* $k¢$, $k \geq 8$, *can be made up from* $3¢$ *and* $5¢$ *stamps.*

BASE CASE. One $5¢$ and one $3¢$ stamps make $8¢$ postage. This completes the base case.

INDUCTION STEP. Assume any postage of up to $k¢$ can be made up from $3¢$ and $5¢$ stamps, consider making $(k + 1)¢$ postage. There are two cases, depending on whether a $5¢$ stamp is used to make $k¢$ postage.

CASE I If a $5¢$ stamp is used to make $k¢$ postage then replacing that stamp with two $3¢$ stamps makes $(k + 1)¢$ postage.

> CASE II If k¢ postage is made up of 3¢ stamps only, then there must be at least three 3¢ stamps because $k \geq 8$. Replacing three of the 3¢ stamps with two 5¢ stamps then makes $(k+1)$¢ postage.

The two cases complete the induction case and the proof of the claim. □

5.3 "Strong" Induction

Sometimes, it is convenient to use a form of inductive argument in which the hypothesis is assumed to hold for all numbers *up to k*. In words, the *"strong" induction* principle says.

> STRONG INDUCTION
>
> *If you can prove*
>
> (BASE CASE) $H[1]$ holds.
>
> *and*
>
> Whenever $H[1]$, $H[2]$, ..., $H[k-1]$ all hold, so does $H[k]$.
>
> *then you may conclude by induction that*
>
> $H[n]$ holds for all $n \in \mathbb{W}$.

This principle is equivalent to the first principle because the induction hypothesis is still a proposition about a number k:

$$H'[k] \equiv \text{"If for all } j < k,\ H[j] \text{ holds then } H[k] \text{ also holds."}$$

In fact, Example 5.4 does just that. See also Exercise 7.

REMARK. Most presentations of strong induction omit the explicit requirement of a base case. It does not need to be explicit because logic implicitly forces us to prove one. The premise in $H'[1] \equiv$ *If for all whole numbers $j < 1$ $H[j]$ then ...,"* is vacuously true, so $H'[1]$ logically reduces to $H[1]$, which must be proven as a separate (base) case.

Example 5.5 Proposition. *Every number greater than 2 is uniquely decomposable into a product of prime numbers.*

PROOF: The proof is by (strong) induction on k with hypothesis

> $H[k] \equiv$ all numbers less than $k > 2$ are uniquely decomposable into a product of primes.

INDUCTION. Assume that every number less than k is uniquely decomposable. If k is prime, then it not further dcomposable. Otherwise, k is a composite number,

$$k = st, \text{ where } s < k \text{ and } t < k$$

By the induction hypothesis, s and t are uniquely decomposable,

$$\begin{array}{rcl} s & = & p_1 \cdot p_2 \cdot \cdots \cdot p_n \\ t & = & q_1 \cdot q_2 \cdot \cdots \cdot q_m \end{array}$$

Hence,

$$k = s \cdot t = (p_1 \cdot p_2 \cdot \cdots \cdot p_n) \cdot (q_1 \cdot q_2 \cdot \cdots \cdot q_m)$$

and this decomposition is unique, up to rearranging the p_is and q_js. This completes the induction conclude that all numbers are uniquely decomposable into primes.

□

Exercises 5.3

1. Prove that for all whole numbers n,

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

2. Prove that for all $n \in \mathbb{N}$,

$$\sum_{i=0}^{n} i(i!) = (n+1)! - 1$$

3. Prove that for all whole numbers n, 6 evenly divides $n^3 - n$.

4. Prove that for all whole numbers $n > 4$, $2^n > n^2$.

5. Prove that for all natural numbers n, $\sum_{i=0}^{n} i^3 = \left(\sum_{i=0}^{n} i\right)^2$.

6. The *"choose"* function is defined

$$\binom{s}{r} = \frac{s!}{r!\,(s-r)!}$$

It is the number of different ways to choose r objects from a set of s objects. Prove that for all $n \in \mathbb{N}$,

$$\sum_{i=0}^{n-1} (2i)^2 = \binom{2n}{3}$$

7. Explain how the proposition *for all* $n \in \mathbb{W}$ *such that* $n > a$, $H(n)$ can be transformed to an equivalent proposition, *for all* $n \in \mathbb{W}$, $H'(n)$. Perform this transformation on the proposition in Exercise 4

8. Repeat the proof of Example 5.5 using the first Principle of Induction, with the induction hypothesis:

$H(k) \equiv$ For all $j \leq k$, j is uniquely decomposable into a product of primes

5.4 More Examples of Induction

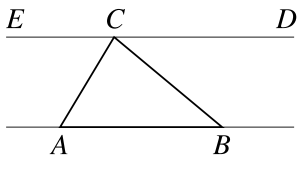
Although numerical induction proofs can always be reduced to arguments about a number, induction is not used only to prove facts about arithmetic. In this section, we look at a few non-arithmetic results. The first has to do with geometry.

Example 5.6 A polygon is *convex* if every line joining the points of the polygon lies within the polygon. In other words, all vertices of a convex polygon "point out." Prove that the sum of the interior angles of a n-sided convex polygon, $n \geq 3$, is equal to $(n-2) \cdot 180°$.

PROOF: The proof is by induction, with hypothesis

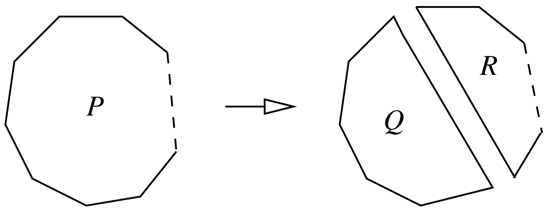
$H[k] \equiv$ The sum of the interior angles of a convex polygon fewer than k sides, $k \geq 3$ is $(k-2) \cdot 180°$.

(BASE CASE) In the base case, $H[3]$ asserts that the sum of the angles of a triangle is $(3-2) \cdot 180° = 180°$. We learned in elementary geometry that this is the case. The proof, if you saw one, involved placing the triangle between two parallel lines, as shown below, and using the axioms of Geometry. Since lines ED and AB are parallel, $\angle ACE = \angle CAB$ and $\angle DCB = \angle CBA$. Furthermore, since ECD is a straight line, $\angle ECA + \angle ACB + \angle DCB = 180°$.



(Check that the argument doesn't depend on ABC being an acute triangle!)

(INDUCTION STEP) Assume $H[k]$. Now consider any polygon P with $k+1$ sides. Since P has at least 4 sides, it is possible to divide it into two convex polygons by drawing an interior line between two of its vertices:



Call these two polygons Q and R. Now, both Q and R are convex and have fewer sides than P. In particular, each has *at most* k sides. Suppose Q has $l_Q \leq k$ sides and R has $l_R \leq k$ sides. Together, Q and R have two more sides than P; that is,

$$l_Q + l_R - 2 = k + 1 \tag{5.1}$$

In addition, since P is formed by putting Q and R side-by-side: The sum of P's interior angles is equal to the sum of the interior angles of Q and R. By the induction hypothesis, the sum of the interior angles of Q is $(l_Q - 2) \cdot 180°$. By the induction hypothesis the sum of the interior angles of R is $(l_R - 2) \cdot 180°$. Therefore, the sum of P's interior angles is

$$\begin{aligned} & ((l_Q - 2) + (l_R - 2)) \cdot 180° \\ &= ((l_Q + l_R - 2) - 2) \cdot 180° && \text{(rearrange sums)} \\ &= ((k + 1) - 2) \cdot 180° && \text{(by Equation 5.1)} \end{aligned}$$

This completes the induction and proves that the sum of the interior angles of any convex polygon with n sides is $(n - 2) \cdot 180°$.

□

Remark. There are three points of interest about Example 5.6.

1. The statement has the form: *For all* $n \geq 3$, $H[n]$, so the base case is translated to $k = 3$. The hypothesis could be changed to make 1 the base case:

   $H'(n) \equiv$ *The sum of the interior angles of a convex polygon with at most* $k + 2$ *sides is* $k \cdot 180°$.

2. Second, had the form of the proof been a strong induction, the hypothesis could have been written:

   $H'(n) \equiv$ *The sum of the interior angles of a convex polygon with* k *sides is* $(k - 2) \cdot 180°$.

   The missing phrase "at most" becomes part of the proof scheme.

3. In this proof the induction hypothesis is used *twice* to reason about polygons Q and R.

□

In the next example, we prove a distributive law for the operations of intersection and union on sets. A number of similar laws are proven as exercises.

Example 5.7 *Let $B, A_1, A_2, \ldots,\ A_n$, $n \geq 1$, be any collection of sets.* Then

$$H[n] \ \equiv\ B \cap \bigcup_{i=1}^{n} A_i = \bigcup_{i=1}^{n} B \cap A_i$$

□

Proof by induction on k with hypothesis $H[k]$.

(base case) If there is just one set A_1,

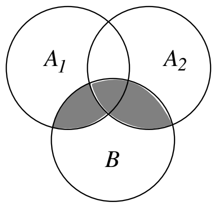
$$B \cap \bigcup_{i=1}^{1} A_i = B \cap A_1 = \bigcup_{i=1}^{1} (B \cap A_i)$$

Although this proves the base case, it is worthwhile looking at the case for two sets.

Lemma I

$$B \cap (A_1 \cup A_2) = (B \cap A_1) \cup (B \cap A_2)$$

Proof. This Venn diagram should suffice:

$B \cap (A_1 \cup A_2) =$  $= (B \cap A_1) \cup (B \cap A_2)$

□

(induction step) Assume $H[k]$ holds. For a collection of sets $A_1, A_2, \ldots, A_{k+1}$, consider

$$B \cap \bigcup_{i=1}^{k+1} A_i$$

The extended union can be expanded once to

$$= B \cap \left(\bigcup_{i=1}^{k} A_i \cup A_{k+1} \right)$$

By Lemma I above, this intersection can be

$$= \left(B \cap \bigcup_{i=1}^{k} A_i \right) \cup (B \cap A_{k+1})$$

According to the induction hypothesis, intersection with B distributes over the extended union:

$$= \bigcup_{i=1}^{k} (B \cap A_i) \cup (B \cap A_{k+1})$$

Increasing the upper bound of the extended union, we get

$$= \bigcup_{i=1}^{k+1} (B \cap A_i)$$

This completes the induction step and the proof of the claim.

Exercises 5.4

1. Prove that for all natural numbers $n \geq 1$, if $B, A_1, \ldots, A_n$ are sets, then

$$B \cup \bigcap_{i=1}^{n} A_i = \bigcap_{i=1}^{n} (B \cup A_i)$$

2. For $n \in \mathbb{N}$, let Q, P_1, $P_2, \ldots, P_n$ be propositions. Prove the following:

(a) $Q \wedge \bigvee_{i=1}^{n} P_i = \bigvee_{i=1}^{n} (Q \wedge P_i)$ (c) $Q \vee \bigwedge_{i=1}^{n} P_i = \bigwedge_{i=1}^{n} (Q \vee P_i)$

(b) $\neg \bigwedge_{i=1}^{n} P_i = \bigvee_{i=1}^{n} (\neg P_i)$ (d) $\neg \bigvee_{i=1}^{n} P_i = \bigwedge_{i=1}^{n} (\neg P_i)$

3. Prove Example 5.6 using the first principle of numerical induction, with induction hypothesis

$H(k) \equiv$ *"The sum of the interior angles of any convex polygon $\underline{\text{with exactly } k \text{ sides, } k > 3,}$ is $(k-2) \cdot 180°$."*

4. Recall that a *tree* is a finite acyclic graph with a single root node of in-degree 0 and all of whose nodes except the root have in-degree 1. Prove that the number of nodes in a binary tree is exactly one greater than its number of edges.

5. Recall (Exercise 2.3-2) that a *binary tree* is a tree whose interior nodes all have out-degree 2. Prove that the number of leaves of a binary tree is exactly one greater than the number of interior nodes.

6. What, if anything, is wrong with

> **Proposition.** *All horses are the same color.*
>
> Proof. *At any time, the number of horses on Earth is finite. So it suffices to show that any herd (finite set) of horses are all the same color. This we prove by induction with hypothesis*
>
> $H[k] \equiv$ *"Any of herd fewer than k horses are all the same color"*
>
> (base case) *All horses in a herd of one horse are the same color.*
>
> (induction step). *Assume that all the horses in a herd of fewer than k horses are the same color. Now, take any herd of k horses. Remove one horse and call it H_1. The remaining horses form a herd of $k - 1 < k$ horses, and by the induction hypothesis, these are all the same color. Take a horse from the remaining herd, call it H_2, and put it together with H_1 to form a herd of 2 horses. Again by the induction hypothesis H_1 and H_2 are the same color. Hence H_1 is the same color as every horse in the original herd. This completes the induction step and concludes the proof.* □

7. In the game of *Nim*, there are two players and two piles of match sticks. Players alternate, each removing some number (non-zero) of matches from one of the piles. The player who removes the last match wins. Prove:

> Claim. *In a game of Nim, if at one player's turn, if the two piles contain an equal number of matches, the other player can always win.*

5.5 Loop Invariants

With the next two examples, we embark on a study of rigorous reasoning about programs. The first step is a small one:

Example 5.8 Consider the following program, $\mathcal{P}$, in which program variables w, x, y and z range over integers.

```
P:  begin
    { $A > 0$ }
    $x$ := $A$;
    $y$ := $B$;
    $z$ := 0;
ℓ:  while $x \neq 0$ do
       begin
       $x$ := $x - 1$;
       $z$ := $z + y$
       end;
    $w$ := $z$;
    { $w = A \times B$ }
    end
```

The comments indicate that if A is nonnegative, this program computes the product $A \times B$, leaving the result in w. An intuitive explanation is that the `while` statement ℓ "loops" x times, adding y to z each time through. Of course, this only works if x is a nonnegative value. Induction may be used to make this explanation more rigorous:

Proposition A. *If program $\mathcal{P}$ ever reaches while-loop ℓ with $x \in \mathbb{W}$, then $\mathcal{P}$ halts with $w = z + xy$.*
Before proving the proposition, observe that if it holds, then we can conclude

Corollary B. *$\mathcal{P}$ computes $A \times B$.*
the initial assignments, program variables x, y, and z hold values A, B, and 0, respectively, so that the program terminates with $w = 0 + A \times B = A \times B$ $\square$

The PROOF of Proposition A is by induction on $n \in \mathbb{N}$ with hypothesis

$H[n] \equiv$ *If $\mathcal{P}$ ever reaches while-loop ℓ with $x = n$, then $\mathcal{P}$ halts with $w = z + xy$.*

BASE CASE, $H[0]$. Should $\mathcal{P}$ reach ℓ with $x = 0$ the test "$x \neq 0$" fails; so the program immediately leaves the loop and sets w to $z = z + 0y = z + xy$.

INDUCTION STEP. Assume $H[k]$ and now suppose that $\mathcal{P}$ reaches ℓ with $x = k + 1$. Then the test "$x \neq 0$" succeeds; so $\mathcal{P}$ executes the body of the loop and returns to ℓ with new values in the three program variables, given by:

$$\begin{aligned} x' &= x - 1 = k \\ y' &= y \qquad \text{(y is unchanged)} \\ z' &= z + y \end{aligned}$$

But since program variable x now contains k the induction hypothesis applies. Hence, $\mathcal{P}$ halts with $w = z' + x'y'$. That is

$$\begin{aligned} w &= z' + x'y' \\ &= z + y + (x - 1)y \\ &= z + y + xy - y \\ &= x + xy \end{aligned}$$

This concludes the induction step and the proof. $\square$

REMARK. In the statement $H[n]$ there is confusion between about *when* a program variable contains a particular value. The equation "$w = z + xy$" refers to the value in w *after* the loop executes and the values of x, y, and z *before* the loop starts executing. This aspect of time makes programs hard to talk about; it is caused by the lack, at this point, of a good mathematical model of how programs work. $\square$

The proof of Proposition A in Example 5.8 hinges on discovering a property that holds *each time the program returned to the beginning of its loop*. Such a property is called a loop's *invariant*. It "captures the essence" of what the loop does. Generalizing the argument to a principle for reasoning about loops yields us a new form of induction.

Theorem 5.1 *[Theorem on Loop Invariants] In the* STMT *programming language of Section 1.4, let B be any test, S any statement, and consider the program fragment*

```
       ⋮
ℓ:  while  B do { I }  S;
ℓ′:    ⋮
```

Suppose I is an assertion with the property that, whenever both I and B hold before S executes, I holds again after S executes. Then if $\mathcal{P}$ reachs ℓ with I true, and should $\mathcal{P}$ ever reach ℓ', I will be true and B will be false.

PROOF: The proof is by induction, of course. It uses the fact that if $\mathcal{P}$ reaches the point ℓ' it must do so by executing the body S some finite, integral number of times—the STMT programming language has no `goto` statements or loop escapes. Hence, the induction hypothesis is

$H[k] \equiv$ *Should $\mathcal{P}$ ever reach ℓ with I true, and if $\mathcal{P}$ then reaches the point ℓ' after going through the loop $\underline{\text{exactly } k \text{ times}}$, I will be true and B will be false.*

Completing the proof is a simple exercise. □

Example 5.9 The Theorem on Loop Invariants allows us to distill the proof of Proposition A to its essential details, once we have discovered the invariant property, $I \equiv A \times B = z + xy$. The theorem takes care of the details of a numerical induction so we don't have to repeat them.

The PROOF distills to

(INITIALIZATION) By looking at the initial assignments we see that program $\mathcal{P}$ first reaches the loop with

$$z + xy = 0 + A \times B = A \times B$$

(INVARIANCE) Suppose that $\mathcal{P}$ reaches ℓ with $x + xy = A \times B$ and $x \neq 0$. The body of the loop executes and $\mathcal{P}$ returns to ℓ with new values given by:

$$\begin{array}{rcl} x' & = & x - 1 = k \\ y' & = & y \\ z' & = & z + y \end{array}$$

Now, $x'y' + z' = (x-1)y + (z+y) = xy - y + z + y = xy + z = A \times B$. Thus, the invariant still holds.

Therefore, by the Theorem on Loop Invariants, when the program leaves its loop, we will have both $A \times B = z + xy$ and $x = 0$; hence $w = z = A \times B$. □

Exercises 5.5

1. Complete the proof of Theorem 5.1.

2. Consider the program

```
$\mathcal{P}$: begin
    {$A$, $B > 0$}
    $x := A$;
    $y := B$;
$\ell$: while $x \neq 0$ do
        begin
        $x := x - 1$;
        $y := y + 1$
        end;
    end {$y = A + B$}
```

Use Theorem 5.1 and invariant assertion

$$I \equiv x + y = A + B$$

to prove that this program computes $A + B$.

3. Consider the program

```
$\mathcal{P}$: begin
    {$A$, $B > 0$}
    $q := 0$;
    $r := A$;
$\ell$: while $r \geq B$ do
        begin
        $q := q + 1$;
        $r := r - B$
        end;
    end {$A = qB + r \wedge r < B$}
```

Use Theorem 5.1 and invariant assertion

$$I \equiv A = qB + r$$

to prove that this program computes the quotient and remainder of A and B.

Chapter 6

Program Analysis, Order and Countability

This chapter begins with a discussion of program performance analysis, the problem of estimating how much time and space a program uses. Exact estimations are challenging and not always useful, particularly for comparing programs.

It makes sense to use *approximate analyses*, classifying programs with peformance *bounding* functions. Bounding approximations are compared and combined using *order arithmetic*, described Section 6.2.

Section 6.3 we return to the idea of counting to consider *cardinalities* of infinite sets. The techniques used in reasoning about *countability* also come into play when reasoning about *computability* and *complexity* of problem solving algorithms. The chapter ends with a brief survey of these results.

6.1 Performance Estimation Example

A program's performance may be measured by its *running time* and its *space consumption*. Here we consider only running time, analyzing a sorting algorithm as a running example. For the purpose of the analysis, let us add three new forms to the STMT programming language of Section 1.4:

(a) Add a statement `skip` , which does nothing.

(b) Add *arrays*. `A[`ℓ `..` u`]` denotes an array, indexed from *lower bound* ℓ to *upper bound* u. An array may not be accessed outside these bounds. The expression `A[`i`]` refers to the array `A` 's content at index i, $\ell \leq i \leq u$..

(c) Add a `for-do` statement,

$$\texttt{for}\ i\ \texttt{from}\ \ell\ \texttt{to}\ u\ \texttt{by}\ k\ \texttt{do}\ \textsc{stmt}$$

where ℓ, u and k are numerical terms. A `for-do` statement expands to the `while-do` statement below. Its form allows us to more easily count the number of loop iterations.

```
begin
i := ℓ;
while i ≤ u do begin STMT; i := i + k end
end
```

6.1.1 The Program

The program below *sorts* an array, $A[1..N]$, of numbers. That is, it places those elements in numerical order. The program implements a simple sorting algorithm called *SelectionSort*, which works by repeatedly finding the smallest number left in the remaining array and moving it toward the front. It is not necessary to understand this algorithm to estimate its performance, but observe that it is a `for` loop whose body contains another, *nested*, `for` loop.

```
begin
for i from 1 to N by 1 do
   begin
   s := i;
   for j from i+1 to N by 1 do
      if A[j] < A[s])
         then s := j
         else skip;
   t := A[s];
   A[s] := A[i];
   A[i] := t;
   end
end
```

6.1.2 *What Are We Counting?*

Our goal is to measure how much work this `SelectionSort` program does. This kind of analysis can be based on many measures, such time, or memory, or both. At different levels, digital hardware for example, the units of measure might be entirely different.

In essence, this example is estimating execution *time*, but not real elapsed time. We shall count how many elementary operations the program performs. So this is yet another kind of counting problem. Let us proceed by working from the inside out.

(a) The innermost conditional statement,

```
if A[j] < A[s] then s:=j else skip
```

performs a less-than test and, if it succeeds, an assignment statement. Let us count this as two units of work. If the test fails, the program performs a `skip`, and thus performs only one unit of work for the test. Since we cannot know which of these branches will execute, we make the *worst-case* estimate that the conditional performs 2 units of work.

(b) The inner `for-do` loop iterates from $j = i+1$ to $j = N$, doing (at most) 2 units of work on each iteration. The inner loop starts by assigning the initial value, $i+1$ to the index variable j. Then it tests $j \leq N$ for loop termination. On each iteration, it performs the conditional statement, and then increments j. The sum of all operation in the inner loop is thus

$$1 + \sum_{j=i+1}^{N} [1+2+1] + 1$$

The number of steps from $i+1$ to N, is $N-(i+1)+1 = N-i$, so the total is

$$1 + \sum_{j=i+1}^{N} 4 + 1 \;=\; 4(N-i)+2$$

units of work.

(c) Similarly, the outer loop performs the inner loop N times with four additional assignment statements; so it performs

$$\begin{aligned}
& 1 + \sum_{i=1}^{N} [1 + [4(N-i)+2] + 3] + 1 \\
= \;& 2 + \left[\sum_{i=1}^{N} 4(N-i) + 6\right] \\
= \;& 2 + 4\sum_{i=1}^{N} N - 4\sum_{i=1}^{N} i + 6\sum_{i=1}^{N} 1 \\
\overset{!?}{=} \;& 2 + 4N^2 - 4\left[\frac{N(N+1)}{2}\right] + 6N \\
= \;& 2 + 4N^2 - (2N^2 + 2N) + 6N \\
= \;& 2N^2 + 4N + 2
\end{aligned}$$

units of work. The key step in the derivation is using Example 5.1 to simplify the middle summation.

6.1.3 Estimation gets a lot harder

In this example, we were able to easily determine the number of iterations hrough both loops. `While` loops and `do` loops sometimes have more complicated index-variable manipulations that make this determination difficult. We were lucky to be able to find the simplification $\sum_{i=0}^{N} = N(N+1)/2$, which allowed all summations to be removed.

In making worst-case assumptions about conditional and repetition statements, our performance estimate is "safe": we can say that the `SelectionSort` program performs *no more* than $2N^2+4N+2$ units of work. A more accurate estimate would involve statistical models of data, such as "How likely is it that the conditional will perform a `skip` ?"

Exercises 6.1

1. Estimate the performance of the *Bubble Sort* program

```
begin
for $i$ from 1 to $N-1$ by 1 do
   begin
   for $j$ from 1 to $i-1$ by 1 do
      if $A[j] \leq A[j+1]$
      then skip;
      else
        begin
        $t$ := $A[j]$ ;
        $A[j]$ := $A[j+1]$ ;
        $A[j+1]$ := $t$
        end
   end
end
```

6.2 Order Notation and Order Arithmetic

Functions from $\mathbb{N}$ to $\mathbb{R}$ are used to estimate program running times, memory requirements, file sizes, and other resources that may vary depending on some program parameter. The rate at which these resources grow as the problem size gets larger a way to characterize an algorithm for solving the problem. Often, this rate cannot be precisely determined, of if it can, the formula is too complicated to work with.

The example in Section 6.1 estimates that *SelectionSort* performs $2N^2+4N+2$ basic operations, where N is the size of the array to be sorted. For large values of N the estimate is dominated by the $2N^2$ term:

N	$\dfrac{2N^2}{2N^2+4N+2}$
10	0.826446280...
100	0.980296049...
1,000	0.998002996...
10,000	0.999800029...
100,000	0.999980000...
1,000,000	0.999998000...
10,000,000	0.999999800...
100,000,000	0.999999980...

As N grows, the contribution of the $4N+2$ becomes negligible. For the purpose of *classification* we can say that the performance of *SelectionSort* "varies with twice the square of the array size."

In practice, if one program runs two times faster than another, it is a substantial improvement. On the other hand, computers get twice as fast every few years, so a factor of two is not significant from a classification standpoint. If we could improve *SelectionSort* to run in $(1.5)N^2 + 9N + 30$ time, it would run in time proportional to N^2. However, if we could find a program whose performance varied at an $N \log N$ rate, that would be theoretically significant, even if the program was slower for small values of N.

These ideas are captured in the next definition, in which $\mathbb{R}^+$ stands for the set of postitive real numbers, $\{x \in \mathbb{R} | x \geq 0\}$.

Definition 6.1 *Let* $f, g\colon \mathbb{N} \to \mathbb{R}^+$. *We say that* f is of order g, *written* $f \in O(g)$, *if there exist* $N \in \mathbb{N}$ *and* $C \in \mathbb{R}$ *such that for all* $n \leq N$, $f(n) \leq C \cdot g(n)$.

N and C are sometimes called *witnesses*. N may be thought of as a *threshold* and C as a *proportionality constant.*

Example 6.1 Let $f(n) = 2n^2 + 5n + 3$ and $e(n) = n^2$. Show that $f \in O(e)$.

SOLUTION: We need to find a *constant factor*, C and a *threshold* value N at which

$$\text{for all } n \geq N,\ 2n^2 + 5n + 3 \leq Cn^2$$

Consider each term in $f(n)$ to find a dominating value in terms of n^2:

(a) If $5 \leq m$ then $5m \leq m^2$.

(b) If $2 \leq m$ then $3 \leq m^2$.

(c) To satisfy both (a) and (b) take N to be 5. Then for any $n \geq N$

$$f(n) = 2n^2 + 5n + 3 \leq 2n^2 + n^2 + n^2 = 4n^2$$

Thus $f \in O(e)$ with witnesses $N = 5$ and $C = 4$.

□

"Big Oh" relationships may be expressed without giving names to the functions, by just using their defining expressions. So Example 6.1 might have been written,

$$\text{Show that } 2n^2 + 5n + 3 \in O(n^2)$$

The propositions and the exercises that follow state some basic properties about O.

Proposition 6.1 *If $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$.*

PROOF: Exercise 3

□

Proposition 6.2 *If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$ then $f_1(n) + f_2(n) \in O\big(g_1(n) + g_2(n)\big)$.*

PROOF: Exercise 6

□

Proposition 6.3 *If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$ then $f_1(n) \times f_2(n) \in O\big(g_1(n) \times g_2(n)\big)$.*

An example of a property for which there is no simple relationship is function composition. If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$ we would wish for simple order relationship between $f_1 \circ f_2$ and $O(g_1 \circ g_2$. Unfortunately, there is no simple formula for this relationship.

PROOF: Exercise 7

□

Exercises 6.2

1. For positive real coefficients a_0, a_1, …, a_n, let $f\colon \mathbb{N} \to \mathbb{R}$ be a polynomial function,

$$f(x) = a_n x^n + \cdots + a_1 x^1 + a_0 x^0$$

For example, one such function is

$$f(x) = 3x^4 + 2.0x^3 + 3.3x^2 + 9x + 1$$

in which the coefficients are $a_0 = 1$, $a_1 = 9$, $a_2 = 3.3$, $a_3 = 2.0$, and $a_4 = 3$.

Show, in general, that $f \in O(x^n)$.

2. Define two functions, f and g, for which $f \notin O(g)$ and $g \notin O(f)$.

3. Prove Proposition 6.1: If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$ then $f_1(n) + f_2(n) \in O\big(g_1(n) + g_2(n)\big)$.

4. Prove that for any base b, $\log_b(x) \in O(x)$.

5. Prove that for any bases b and b', $\log_b(x) \in O(\log_{b'}(x))$.

6. Prove Proposition 6.2

7. Prove Proposition 6.3: *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then*

$$f_1(n) \times f_2(n) \in O\big(g_1(n) \times g_2(n)\big)$$

.

8. (Hard) It is a fact that $n^x \in O(2^n)$. Explain or prove this result.

9. Is $2^n \in O(n!)$ or is $n! \in O(2^n)$?

6.3 Cardinality and Countability

For infinite sets, it makes little sense to say $|S|$ is a *number* or that S has a "size". Nevertheless, we are interested in *comparing* infinite sets, so the concept of a set's size must be made more general.

Definition 6.2 *Two sets, S and T, said to be of the* same cardinality *if one can exhibit a one-to-one correspondence between them, that is, if a bijection $f\colon A \leftrightarrow B$ exists.*

In the case of two finite sets, we need only show that $|A| = |B|$. It should also make sense—although we do not yet have the technical means to prove it—that if $|A| \neq |B|$, there can be no bijection between them: no mapping from the larger to the smaller set can be injective. In other words, any function from the larger set to the smaller must map two or more elements to the same target.

Fact 6.4 (Pigeon-Hole Principle) *If $|A| > |B|$ and $g\colon A \to B$, then there are at least two distinct elements, $x \neq y$ in A for which $g(x) = g(y)$.*

The Pigeon-Hole Principle often arises in proofs-by-counter-example, as for example, in showing that a path in $R \subseteq A \times A$ of length $n > |A|$ must contain a cycle.

In the case of infinite sets, we are particularly interested in those having the same cardinality as $\mathbb{W}$. In essence, these are sets that can be indexed by whole numbers, and so can be can be listed "in order,"

$$S = \{s_1, s_2, s_3, \ldots, s_i, \ldots\}$$

For most purposes, it is enough to know that S is "small enough" to be listed in a linear order, even if some of the indices are missing. We don't need a bijection; an injection is good enough.

Definition 6.3 *A set S is* countable *if there exists an injection, $C\colon S \to \mathbb{W}$.*

Proposition 6.5 $\mathbb{N}$ *is countable.*

PROOF: The mapping $C\colon n \to n+1$ gives a bijection, hence also an injection from $\mathbb{N}$ to $\mathbb{W}$.

$\square$

Proposition 6.6 *The integers, $\mathbb{Z}$, are countable.*

PROOF: The proof is done by exhibiting a function that satisfies Definition 6.3. Define $C\colon \mathbb{Z} \to \mathbb{N}$ according to:

$$C(x) = \begin{cases} 2x & \text{if } x \geq 0 \\ -2x-1 & \text{if } x < 0 \end{cases}$$

C maps positive integers to even numbers and negative integers to odd numbers: It is injective; different numbers get different indices. Thus, $\mathbb{Z}$ can be listed

$$\{0, -1, 1, -2, 2, -3, \ \ldots\}$$

$\square$

The next proposition allows us to use any countable set to index another countable set. In other words, we don't have to be overly specific about how the elements are ordered, because in most cases we are more interested in cardinality, not the exact correspondence.

Proposition 6.7 *A set S is countable if there exists an injection from S to a countable set, A.*

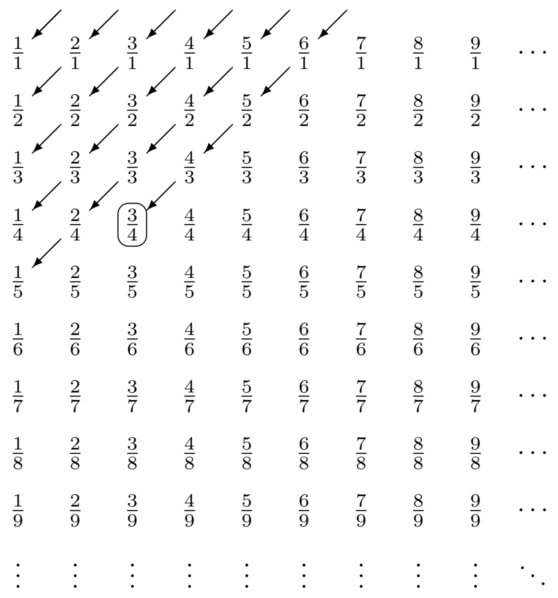
PROOF: If $f\colon S \to A$ and $C\colon A \to \mathbb{W}$ are both injections, then so is $f \circ C\colon S \to \mathbb{W}$ (Exercise 2.1-9).

$\square$

Theorem 6.8 *The rational numbers, $\mathbb{Q}$, are countable.*

PROOF: Recall that $\mathbb{Q} = \{\frac{n}{m} \mid n, m \in \mathbb{Z} \text{ and } m \neq 0\}$. For the moment, let us consider only the rationals with positive numerators, $\mathbb{Q}^+ = \{\frac{n}{m} \mid n, m \in \mathbb{W}\}$. We can use **??** later to extend the result to $\mathbb{Q}$.

Consider $\mathbb{Q}^+$ layed out as an infinite two-dimensional array, shown below. We will define the required bijection C so that it orders this array along successive diagonals.



To calculate $C(\frac{x}{y})$ look at Figure 6.1 and consider:

(a) Along the diagonal containing $\frac{x}{y}$ adding the "numerator" and "denominator" always results in the same number. For instance, along the diagonal containing $\frac{3}{4}$, $6+1=5+2=4+3=3+4=2+5=1+6=7$. Moreover, $x+y$ is two greater than the length preceeding diagonal.

(b) All the rationals above *this* diagonal are already accounted for, and that number is the sum of the preceeding diagonals's lengths, $1+2+\cdots+(x+y-2)$. Recall from Example 5.1 that $\sum_{i=1}^{n} = n(n+1)/2$, so

$$\sum_{i=1}^{x+y-2} i = \frac{(x+y-2)(x+y-1)}{2}$$

(c) The number of points along the diagonal leading to $\frac{x}{y}$ is $y-1$.

Adding these together, we get

$$\begin{aligned}
C(\tfrac{x}{y}) &= \frac{((x+y)-1)((x+y)-2)}{2} + y - 1 \\
&= \frac{(x+y)^2 - 3(x+y) + 2}{2} + \frac{2y-2}{2} \\
&= \frac{(x+y)^2 - 3x - 3y + 2 + 2y - 2}{2} \\
&= \frac{(x+y)^2 - 3x - y}{2}
\end{aligned}$$

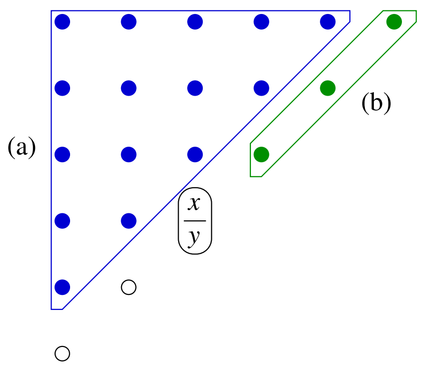


Figure 6.1: Calculation of $C(\frac{x}{y})$

$\square$

REMARK 1. The display leaves out rationals of the form $\frac{0}{y}$, which could have been included by adding another row along the top and adjusting the formula for C. This is left as an Exercise. $\square$

REMARK 2. When one thinks of *rational numbers*, one ordinarily thinks of the real numbers they represent and also includes negative values. These could be incorporated in C, for instance, by mapping positive rationals to even numbers and negative rationals to odd numbers, as in Proposition 6.6. $\square$

REMARK 3. You might be concerned about the fact that C maps "equivalent" rationals to different values. For instance $C(\frac{1}{1}) = 1$, $C(\frac{2}{2}) = 5$, $C(\frac{3}{3}) = 13$, $C(\frac{4}{4}) = 25$, and so forth, but all these fractions reduce to the number 1. In this theorem we are counting *numerals*, not the *numbers* they may represent. Even if we were to consider equivalent values to represent the same number, countability just says that there are *no more* elements of a set S than there are whole numbers, so it is all right to count something more than once, as long as everything is counted at least once. $\square$

In essence We have actually shown,

Proposition 6.9 $\mathbb{W} \times \mathbb{W}$ *is countable.*

PROOF. The proof uses the same argument as that of Theorem 6.8 to define a counting function $C\colon \mathbb{W}^2 \to \mathbb{W}$. $\square$

If follows by induction that

Corollary 6.10 *For all* $n \in \mathbb{W}$,

$$\mathbb{W}^n = \overbrace{\mathbb{W} \times \cdots \times \mathbb{W}}^{n\ times}$$

is countable.

The PROOF is left as an exercise. □

The next theorem illustrates an important proof technique called *diagonalization*, used frequently in theoretical computer science.

Theorem 6.11 $\mathbb{R}$ *is not countable.*

PROOF: This is a *proof by contradiction*, in which we assume the result is false—that $\mathbb{R}$ *is* countable—and deduce that it can't be. The proof depends on the fact that any real number can be expressed as a possibly infinite decimal expansion, and conversely, that any decimal numeral represents some real number.[†] For the sake of simplicity we will consider only the numbers in the interval $[0, 1)$. If we can't count these, we certainly can't count the entire set $\mathbb{R}$.

Suppose that we can count the numbers in this interval. Then there must be a way to list them in order, $\{r_1, r_2, r_3, \ldots\}$. Consider the decimal expansions of the r_i:

$$\begin{array}{lcl}
r_1 & = & 0.\ \boxed{d_{11}}\, d_{12}\ d_{13}\ d_{14}\ d_{15}\ d_{16}\ d_{17}\ d_{18} \ldots \\
r_2 & = & 0.\ d_{21}\ \boxed{d_{22}}\, d_{23}\ d_{24}\ d_{25}\ d_{26}\ d_{27}\ d_{28} \ldots \\
r_3 & = & 0.\ d_{31}\ d_{32}\ \boxed{d_{33}}\, d_{34}\ d_{35}\ d_{36}\ d_{37}\ d_{38} \ldots \\
r_4 & = & 0.\ d_{41}\ d_{42}\ d_{43}\ \boxed{d_{44}}\, d_{45}\ d_{46}\ d_{47}\ d_{48} \ldots \\
r_5 & = & 0.\ d_{51}\ d_{52}\ d_{53}\ d_{54}\ \boxed{d_{55}}\, d_{56}\ d_{57}\ d_{58} \ldots \\
r_6 & = & 0.\ d_{61}\ d_{62}\ d_{63}\ d_{64}\ d_{65}\ \boxed{d_{66}}\, d_{67}\ d_{68} \ldots \\
 & & \quad \vdots \qquad\qquad\qquad\qquad \ddots
\end{array}$$

where each $d_{ij} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ Now consider the numeral

$$0\ .\ d'_{11}\ d'_{22}\ d'_{33}\ d'_{44}\ d'_{55}\ d'_{66}\ \ldots\ d'_{kk}\ \ldots$$

In which each digit d'_{kk} is chosen to be *different* than d_{kk} from the list of r_is. The number this numeral represents cannot be in the list because for all $n \in \mathbb{W}$ it differs at the n^{th} decimal place from the n^{th} numeral in the list (hence the term *diagonalization*). This contradicts the assumption that all numbers are listed, so that assumption must be false; and this completes the proof of the theorem. □

REMARK. Although it is essentially correct, this argument is subtly flawed, because the same number can have two different decimal representations. For instance, $0.0999\ldots = 0.1$ which follows from the fact that

$$(10x - x) \div 9 = \frac{9}{9}x = x$$

The proof can be repaired by excluding redundant expansions, but the details are not illuminating. □

[†]Recall the footnote on page 4

Exercises 6.3

1. Modify the proof of Proposition 6.8 to include rationals of the form $\frac{0}{y}$ for $y \in \mathbb{W}$.

2. Prove: *If sets A and B are countable, then so are $A \cup B$, $A \times B$ and $A \setminus B$.*

3. Suppose sets A and B are countable. Explain why $\mathcal{P}(S)$ and $f : A \to B$ are not necessarily countable.

6.4 Decidability

A *decision procedure* is an algorithm that can solve *all instances* of a particular problem.

Chapter 7

Induction II

7.1 Introduction

The sets of interest in computer science tend to be complicated. For instance, consider the problem of defining the set of all legal `C` programs. Of course, the best way to think about such complicated objects is usually to to break them down into simpler pieces. For example, we build a `C` program out of constituent phrases, such as declarations, expressions, assignment statements, and so forth.

The decomposition of a complex set must be systematic. The pieces we break it into must be meaningful, so that we can reason about more complicated objects from facts about their pieces. We want definition methods and reasoning techniques that apply not just to a particular set of objects, but more broadly. We need a general scheme for the mathematical treatment of languages and what they mean. We begin with two examples to motivate these ideas. These examples will be used throughout this section and the next to illustrate the topics.

Example 7.1 The Language L. At this point you may want to review Section 1.3 on lanauges. Suppose we wish to define a language called L of simple product expressions Let V be a set of legal constant symbols and program variables and assume that the symbol '`*` ' is not a member of V. L will consist of words over the alphabet $W = V \cup \{*\}$, such as `120`, `5* x`, and `width* height* 5`.

Constants and variables themselves are simple expressions; and given two expressions u and v, one can build a new expression by concatenating, $u\hat{}*\hat{}v$. Now, if we wish to prove that some word $w \in W^+$ is in L, we should be able to provide a derivation of w according to these two rules:

A. Either $w \in V$, or

B. $w = u * v$ and both u and v have a derivation.

For example, we know that `width*height*5` is a word in L because

1.	`width` $\in L$	rule A
2.	`height` $\in L$	rule A
3.	`width*height` $\in L$	rule B with 1, 2
4.	`5` $\in L$	rule A
3.	`width*height*5` $\in L$	rule B with 3, 4

Notice that this is not the only possible derivation. Conversely, if we wish to prove that $w \notin L$ then we should show that no such derivation can exist.

Of course, L does not capture *all* the arithmetic expressions of a programming language. In a later section we will build a more realistic language. Let us now examine a claim about elements of L:

Claim *The number of constants and variables contained in any word $w \in L$ is exactly one greater than the number of '`*` ' symbols.*

INFORMAL PROOF In the first place, each element of V has just one constant or variable and no '`*` 's. In the second place, if any two words u and v have this property, then so does the word $u{*}v$.

This argument is inductive! The hypothesis is:

$$H(w) \equiv \textit{The number of constants and variables in } w \textit{ is exactly one greater than the number of } \texttt{*} \textit{ 's}$$

In a "base case" we proved $H(v)$ for all $v \in V$. In an "induction step" we prove $H(u)$ and $H(v)$ imply $H(u{*}v)$. However, the argument is not *technically* a mathematical induction because H is a predicate on L rather than $\mathbb{W}$. □

Example 7.2 The Relation R We are going to build a relation $R \subseteq \mathbb{N} \times \mathbb{N}$, starting with the ordered pair $(0,0)$. The next pair in the relation is $(1,2)$. In general, whenever we have a pair $(n,m) \in R$ we may add the pair $(n+1, m+2)$ to R, so we can think of R as the limit of a sequence of sets:

$$\begin{array}{rcl}
R_0 & = & \{(0,0)\} \\
R_1 & = & \{(0,0),(1,2)\} \\
R_2 & = & \{(0,0),(1,2),(2,4)\} \\
 & \vdots & \\
R_k & = & \{(0,0),(1,2),(2,4),\ \ldots,(k,2k)\} \\
 & \vdots & \\
R & = & \bigcup_{i\in\mathbb{N}} R_i = \{(0,0),(1,2),(2,4)\ \ldots,(k,2k),\ \ldots\}
\end{array}$$

We can see in each R_k the derivation of the pair $(k,2k)$. We can also write down an explicit definition of R_k:

$$R_k = \{(x,2x) \mid x \in \mathbb{N} \text{ and } x \le k\}$$

The limit of this construction is

$$R = \{(x, 2x) \mid x \in \mathbb{N}\}$$

In fact, R happens to be a function, so not only can we build languages in this step-by-step way, but we can also define relations and functions. We will see much more of this technique later.

□

7.1.1 The Problem of Self Reference

Consider the riddle

> *In a certain restaurant there works a Waiter who serves every person that does not serve themself.*
>
> QUESTION: *Who serves the Waiter?*
>
> ANSWER: *There is no such restaurant.*

Let W stand for the supposed waiter. The logical problem in this riddle is that, were such a restaurant exist, it would be have to be the case that W *serves* W *iff* W *does not serve* W. Thus, the problem statement itself leads immediately to a contradiction.

The source of the problem may be that the assertion "W *serves* W" does not make sense. It seems to, but not all such self-referential statements do. Consider, for example, this proposition, known as the *Liar's Paradox*:

> *The sentence in this box is false.*

In order to talk about this sentence, it helps to give it a name, say S. Then we can write

$$S \equiv \text{“}S \text{ is false.”}$$

S is a simple declaration of fact; it should be either *true* or *false*, but it cannot be either. The same is true of the "system" of two sentences,

$$\begin{array}{rcl} A & \equiv & \text{“Sentence } B \text{ is true.”} \\ B & \equiv & \text{“Sentence } A \text{ is false.”} \end{array}$$

There is no consistent truth assignment for A and B. Again, self-reference (direct or indirect) lies at the heart of the riddle.

Should we just outlaw self reference? This would be paying too high a price. Consider the language $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$ over the alphabet $\{\mathtt{a}, \mathtt{b}\}$. Taking $\mathtt{a}^n$ to mean a word consisting of n $\mathtt{a}$s, this language consists of all words containing an equal number of $\mathtt{a}$s and $\mathtt{b}$s, with all the $\mathtt{a}$s occuring before any of the $\mathtt{b}$s. In a programmer's manual, languages are often described using a a "grammar," called Backus-Naur notation that looks like:

$$\langle L \rangle ::= \varepsilon \;[]\; \mathtt{a}\hat{}\langle L \rangle\hat{}\mathtt{b}$$

Translated to set notation, the notation says that the language $L \subseteq \{\mathtt{a}, \mathtt{b}\}^*$ is

$$L = \{\varepsilon\} \cup \{\mathtt{a}\hat{\ } w \hat{\ } \mathtt{b} \mid w \in L\}$$

The language description is self-referencing, but certainly makes sense. In fact, describing languages this way is very useful.

When does self-reference make sense and when does it not? As we shall see in this chapter, some kinds of self-referencing inductive and recursive definitions are "sensible" and very useful.

Exercises 7.1

1. Taking the two statements P and Q together, are both true? Are both false? Are they neither true nor false?

$$\begin{array}{ccl} P & \equiv & \text{"Sentence } P \text{ is true or Sentence Q is true."} \\ Q & \equiv & \text{"Sentences } P \text{ and } Q \text{ are not both true."} \end{array}$$

7.2 Inductively Defined Sets

Let us look at what the motivating examples—the language L and the relation R—in the Section 7.1 have in common.

- Both examples involved a set U of values: words in $(V \cup \{\mathtt{*}\})^+$ and pairs in $\mathbb{N} \times \mathbb{N}$, respectively.

- Both constructions started from a "seed" or *base set* from which all elements ultimately are built. For L the base set was V; and for R the base set was $\{(0, 0)\}$.

- Finally, both examples employed a *constructor* function, to make more complicated elements from simpler ones. For L, there was a two-place constructor,

  $$f(u, v) = u \hat{\ } \mathtt{*} \hat{\ } v$$

  For R, the constructor function was $g \colon \mathbb{N}^2 \to \mathbb{N}^2$,

  $$g(n, m) = (n + 1, m + 2)$$

We would like to find properties that tell us exactly what these kinds of sets are, and how to reason about them. Part of the answer lies in the fact that sets like L and R are *closed* with respect to their constructors. This means that if you apply a constructor to elements of the set, the result is also in the set:

Definition 7.1 *If $f \colon U^r \to U$, and $S \subseteq U$, we say S is closed with respect to f iff $s_1, \ldots, s_r \in S$ implies that $f(s_1, s_2, \ldots, s_r) \in S$.*

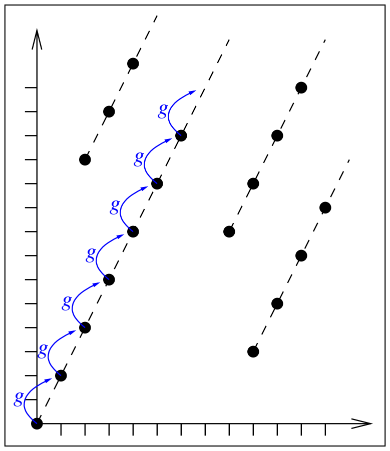
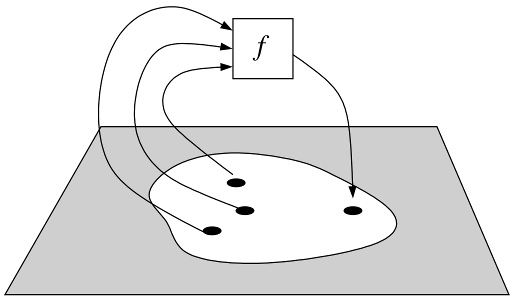


Figure 7.1: A typical subset of $\mathbb{N} \times \mathbb{N}$ containing base set $\{(0, 0)\}$ and closed under the constructor function $g(n, m) = (n + 1, m + 2)$.

Each constructor takes inputs from S to outputs in S.



The example sets L and R that we have been discussing are closed with respect to their constructor functions. If u and v are elements of L then so is the word $u{*}v$. And whenever $(n, m) \in R$, so is $(n + 1, m + 2)$. So closure with respect to the constructor functions is evidently a property of the sets we are building.

However, there may be many subsets of U closed under f. The empty set is closed with respect to every function, vaccuously. The entire set of values, U, is closed with respect to every constructor function, trivially. Hence, the next question to answer is whether we can uniquely determine which closed set a construction gives us.

Another property of the sets of interest is that they are generated by, and therefore contain, the base set. But again, there may be many subsets $S \subseteq U$ which are closed with respect to a constructor f. For example, Figure 7.2 shows the cartesian graph of a typical closed set for the function $g(n, m) = (n + 1, m + 2)$, the constructor function for the relation R. Any collection of half-lines with slope 2 (of course, we are referring only to the discrete points in

$\mathbb{N} \times \mathbb{N}$ which lie on these lines) will be closed under g.

The figure also shows that points "generated" from the base point $(0, 0)$ lie on a single line. As the figure suggests, any closed subset that contains $\{(0, 0)\}$, must also contain the whole of R. In other words, R is the *smallest* closed subset which contains the base set. The property of being smallest determines R uniquely. Let us combine these observations into a definition.

Definition 7.2 *Let U be a set; let $B \subseteq U$. and let $f_1, \ldots, f_m$ be a collection of functions on U. A set A is said to be* inductively defined *from* base set B *and* constructors $f_1, \ldots, f_m$, *if*

(a) *A contains B*

(b) *A is closed with respect to each $f_1, \ldots, f_m$.*

(c) *if S is any subset of U that contains B and is closed with respect to every constructor, then $A \subseteq S$.*

It follows immediately from the definition that inductively defined sets are uniquely defined:

Proposition 7.1 *If A and A' are inductively defined from base set B and functions $f_1, \ldots, f_m$, then $A = A'$.*

Proof: By Definition 7.2(a, b), both sets contain B and are closed under the constructors. If we assume that both A and A' are inductively defined, then by 7.2(c), $A \subseteq A'$ and $A' \subseteq A$. Therefore $A = A'$.

$\square$

The next definition gives us a concise notation for specifying inductively defined sets.

Definition 7.3 (Inductive Set Definition Scheme) *The following language is used to specify an inductively defined set, A:*

1.	$B \subseteq A$	*[A contains the base set]*
2a.	$x_1, \ldots, x_r \in A$ $\Rightarrow f(x_1, \ldots, x_r) \in A$	*[A is closed under constructor f]*
	$\vdots$	
3.	*nothing else is in A*	*[A is the smallest such set]*

Example 7.3 Let us use the definition scheme to specify the language of simple multiplication expressions from the beginning of this section. Let V be a set of constants and program variables. The language $L \subseteq (V \cup \{*\})^+$ is inductively defined according to:

1. $V \subseteq L$
2. $u, v \in L \Rightarrow u{*}v \in L$
3. nothing else

$\square$

Example 7.4 The relation $R \subseteq \mathbb{N} \times \mathbb{N}$ from the beginning of this section is inductively defined according to.

1. $(0,0) \in R$
2. $(n,m) \in R \Rightarrow (n+1, m+2) \in R$
3. n. e.

$\square$

Example 7.5 We can define the *natural numbers* inductively, according to

1. $0 \in \mathbb{N}$
2. $k \in \mathbb{N} \Rightarrow k+1 \in \mathbb{N}$
3. n. e.

$\square$

As the exercises at the end of Section **??** suggest, our definition of inductive sets could be more general than it is. Of particular importance is the idea of simultaneously defining several sets inductively (see Exercise 5). We will see this kind of construction many times in later chapters. It is hard to come up with a *most* general set definition scheme.

7.3 The Principle of Structural Induction.

Let us attempt to use Definition 7.2 to show that the relation R of Example 7.4 is exactly the relation $\{(x, 2x) \mid x \in \mathbb{N}\}$, which we claimed but did not prove at the begining of the last section.

Proposition 7.2 *Let the relation R be defined as in Example 7.4. Define $E = \{(x, 2x) \mid x \in \mathbb{N}\}$. Then $R = E$*

Proof: We will prove that $E \subseteq R$ and $R \subseteq E$. Let g stand for R's constructor function:

$$g(n,m) = (n+1, m+2)$$

Since

$$(0,0) = (0, 2 \cdot 0) \in E$$

and since $(x, 2x) \in E$ implies

$$g(x, 2x) = (x+1, 2x+2) = (x+1, 2(x+1)) \in E$$

Theorem 7.2 says that $E \subseteq R$ because R is the smallest set that contains $(0,0)$ and is closed under g.

But how do we know that $R \subseteq E$? The intuition is pretty clear:

1. R's base element, $(0,0)$, is in E as argued above.

2. a constructed element $(x+1, y+2)$ will be in E only if (x, y) is.

3. The only way to get an element of R is to build it from $(0, 0)$

$\square$

This intuition, generalized, gives us a new Principle of Induction for inductively defined sets:

Theorem 7.3 (Principle of Structural Induction) *Let $A \subseteq U$ be inductively defined from base set B and constructor functions $f_1, \ldots, f_m$. Suppose P is a predicate on U with the following properties:*

(BASE CASE) *For all $x \in B$, $P[x]$ holds.*

(INDUCTION STEP) *For each r-place constructor f,*

$$P[x_1] \wedge \cdots \wedge P[x_r] \quad \Rightarrow \quad P[f(x_1, \ldots, x_r)]$$

Then on may conclude, For all $x \in A$: $P[x]$

PROOF: The proof of the principle is in Section 7.8. $\square$

Recall that we can think of $\mathbb{N}$ as a set defined inductively according to

1. $0 \in \mathbb{N}$
2. $k \in \mathbb{N} \Rightarrow (k+1) \in \mathbb{N}$
3. n. e.

The Principle of Structural Induction, applied to $\mathbb{N}$, says that if you can prove (1) $H(0)$ and (2) $H(k) \Rightarrow H(k+1)$, then you may conclude, *for all* $n \in \mathbb{N}, H(n)$. In other words, the Principle of Structural Induction reduces to ordinary mathematical induction in the case of $\mathbb{N}$

Example 7.6 By using structural induction, we can distill the second half of Proposition 7.2 to its essence. Compare the argument below with the form of a mathematical induction.

> **Claim**. For E and R as defined in Proposition 7.2, $R \subseteq E$.
>
> PROOF. The proof is by structural induction on $x \in R$ with hypothesis $H(x) \equiv x \in E$.
>
> BASE CASE: $(0, 0) = (0, 2 \cdot 0) \in E$

INDUCTION STEP: Suppose $(n, m) \in E$, so that $(n, m) = (x, 2x)$ for some number x. Then

$$g(n, m) = (n + 1, m + 2) = (x + 1, 2x + 2) = (x + 1, 2(x + 1)) \in E$$

By Theorem 7.3 we may conclude, *for all* $x \in R$, $x \in E$. In other words, $R \subseteq E$.

□

Example 7.7 At the beginning of the previous section a language of simple multiplication expressions was defined as follows. Let V be a set of program constants and program variables. Let $W = V \cup \{ * \}$. The language $L \subseteq W^+$ is defined inductively according to:

1. $V \subseteq L$
2. $u, v \in L \Rightarrow u{*}v \in L$
3. n. e.

The following claim was made about elements of L:

> **Claim**: *The number of constants and variables contained in any word* $w \in A$ *is exactly one greater than the number of '* * *'s.*

It was observed that the argument proving this claim was "inductive." We can now recognize it as a structural induction:

PROOF. The proof is by induction on $u \in L$. For the base case, an element of V has just one constant or variable and no '* 's. For the induction step, assume $u, v \in L$ each have this property. That is, u has n '* 's and $n + 1$ symbols from V; and v has m '* 's and $m + 1$ symbols from V. It follows, then, that the word $u{*}v$ has $n + m + 1$ '* 's and $(n + 1) + (m + 1) = (n + m + 1) + 1$ symbols from V.

□

Exercises 7.3

1. Let the set $A \subseteq \mathbb{N}$ be inductively defined according to

1. $1 \in A$
2. $k \in A \Rightarrow k + k \in A$
3. n. e.

Define the set $E = \{2^n \mid n \in \mathbb{N}\}$. *Prove*: $A = E$.

2. Let $V = \{\,\texttt{(}\,,\,\texttt{)}\,\}$ and consider the set L of words in V^+ that is defined inductively according to.

$$\begin{array}{rl} 1. & \texttt{()} \in L \\ 2a. & u \in L \Rightarrow \texttt{(}\,u\,\texttt{)} \in L \\ 2b. & u, v \in L \Rightarrow \texttt{(}\,u\,v\,\texttt{)} \in L \\ 3. & \textit{nothing else} \end{array}$$

Which of the following words are in L?

(a)	`((()))`	(d)	`(()())`
(b)	`()()`	(e)	`(()()())`
(c)	`((()())())`	(f)	`((()()))`

3. Let $V = \{\,\texttt{(}\,,\,\texttt{)}\,\}$ and consider the set L of words in V^+ that is defined inductively according to.

$$\begin{array}{rl} 1. & \texttt{()} \in L \\ 2a. & u \in L \Rightarrow \texttt{(}\,u\,\texttt{)} \in L \\ 2b. & u, v \in L \Rightarrow \texttt{(}\,u\,v\,\texttt{)} \in L \\ 3. & \textit{nothing else} \end{array}$$

Prove: *Every element in L has the same number of* $\boxed{\texttt{(}}$*'s and* $\boxed{\texttt{)}}$*'s.*

4. Think of another property that you can prove about words in the language L of Exercise 2.

7.4 Defining Functions with Recursion

Let $U = \mathbb{N} \times \mathbb{N}$ and define the relation $F \subseteq U^2$ inductively as follows.

1. $(0, 1) \in F$
2. $(n, m) \in F$ implies $(n + 1,\, m(n + 1)) \in F$
3. nothing else is in F

Apparently, F is the factorial function. Let $G = \{(n, n!) \mid n \in \mathbb{N}\}$. Since G contains F's base element, $(0, 1) = (0, 0!)$; and since G is closed under F's constructor because

$$(n, n!) \quad \mapsto \quad (n + 1, n!(n + 1)) = (n + 1, (n + 1)!)$$

we know by Theorem 7.11 that $F \subseteq G$. To prove that $G \subseteq F$, use induction on $k \in \mathbb{N}$ with hypothesis, $H(k) \equiv (k, k!) \in F$. The details are left as an exercise.

We now know that F is a function, and it is customary to write "$F(x) = y$" instead of "$(x, y) \in F$." If we express F's inductive definition in this way, we get:

1. $F(0) = 1$
2. $F(n) = m$ implies $F(n + 1) = m(n + 1)$
3. nothing else

In part 2 the variable 'm' has become just a name for $F(n)$. By using $F(n)$ in place of m, we can further abbreviate the definition to

1. $F(0) = 1$
2. $F(n+1) = F(n) \cdot (n+1)$
3. nothing else

This is a *recursive* definition—F is defined "in terms of itself." It is not a circular definition because F's value for any number execpt 0 is in terms of values for a previous number.

Let us do another example, first in *relational form*, and then again in *functional form*.

Example 7.8 Consider the relation $G \subseteq \mathbb{N} \times \mathbb{N}$ defined inductively as follows:

1. $(0, 0) \in G$
2. $(x, y) \in G$ implies $(x+1,\ y+2x+1) \in G$
3. nothing else

Claim: $(x, y) \in G$ implies $y = x^2$.

PROOF: The proof is by induction on $k \in \mathbb{N}$ with hypothesis, $(k, k^2) \in G$.
BASE CASE: $(0, 0^2) = (0, 0) \in G$ by rule 1.
INDUCTION STEP: Assume that $(k, k^2) \in G$. Then by rule 2,

$$(k+1, x^2+2x+1) = (x+1, (x+1)^2) \in G$$

□
□

Example 7.9 The relation G in the previous exercise is a function, so let us repeat the previous argument, replacing "$(x, y) \in G$" by "$y = G(x)$." It is conventional in recursive definitions to leave out the nothing-else clause.

Consider the function $G\colon \mathbb{N} \to \mathbb{N}$ defined recusively as follows:

1. $G(0) = 0$
2. $G(x+1) = G(x) + 2x + 1$

CLAIM: For all $n \in \mathbb{N}$, $G(n) = n^2$.

PROOF: The proof is by induction on $k \in \mathbb{N}$ with hypothesis, $G(k) = k^2$.
BASE CASE: By the definition of G, part 1,

$$G(0) = 0 = 0^2$$

INDUCTION STEP:

$$\begin{aligned} G(k+1) &= G(k) + 2k + 1 && \text{(G.2)} \\ &= k^2 + 2k + 1 && \text{(Induction Hypothesis)} \\ &= (k+1)^2 && \text{(factoring)} \end{aligned}$$

□
□

7.5 Evaluation of Recursive Functions

Consider the relation $F \subseteq \mathbb{N}^2 \times \mathbb{N}$, defined inductively according to:

$$\begin{array}{rl}
1. & ((0,0),0) \in F \\
2a. & ((n,m),k) \in F \text{ implies } ((n+1,m),k+1) \in F \\
2b. & ((n,m),k) \in F \text{ implies } ((n,m+1),k+2) \in F \\
3. & \text{n. e.}
\end{array}$$

Now suppose we want to find a value ℓ for which $((2,3),\ell) \in F$. In relational form, we can build a "bottom-up" derivation: starting from the base rule:

$$\begin{array}{ll}
 & (\mathit{rule}) \\
\langle\ ((0,0),0) & (1) \\
\ \ ((0,1),2) & (2b) \\
\ \ ((1,1),3) & (2a) \\
\ \ ((2,1),4) & (2a) \\
\ \ ((2,2),6) & (2b) \\
\ \ ((2,3),8)\rangle & (2b)
\end{array}$$

This is not the only way to build the result, and you may wish to convince yourself that $((2,3),8)$ is the only possible value.

F is a function and the functional form of its definition is:

$$\begin{array}{rl}
1. & F(0,0) = 0 \\
2a. & F(n+1,m) = F(n,m)+1 \\
2b. & F(n,m+1) = F(n,m)+2
\end{array}$$

In this form, a derivation of F's value at $(2,3)$ proceeds "top down," for instance,

$$\begin{array}{rll}
F(2,3) & = \text{F(2, 2)} + 2 & (\text{F.2b}) \\
& = (\text{F(2, 1)} + 2) + 2 & (\text{F.2b}) \\
& = ((\text{F(1, 1)} + 1) + 2) + 2 & (\text{F.2a}) \\
& = (((\text{F(0, 1)} + 1) + 1) + 2) + 2 & (\text{F.2a}) \\
& = ((((\text{F(0, 0)} + 2) + 1) + 1) + 2) + 2 & (\text{F.2b}) \\
& = ((((0+2)+1)+1)+2)+2 & (\text{F.1}) \\
& = 8 & (\text{arithmetic})
\end{array}$$

In the relational form, the computation strictly follows the inductive definition of F. The functional form, on the other hand, appears as a set of algebraic identities that describe F; and to evaluate F on an argument, we apply whatever identities we can until we reach an answer. This form may seem more natural—it is closer to our idea of a *computation*—but this intuition it can sometimes lead us into difficulty, as we shall see in the next chapter.

There is a third alternative available to us for this example. It is to observe, perhaps by experimenting with examples, that $F(n,m) = n+2m$. This is easily proved by structural induction on F. In the base case, $F(0,0) = 0 = 0 + 2 \cdot 0$. There are two induction cases,

$$F(n+1,m) \overset{2a}{=} F(n,m) + 1 \overset{IH}{=} (n+2m) + 1 = (n+1) + 2m$$

and

$$F(n,m+1) \overset{2b}{=} F(n,m) + 2 \overset{IH}{=} (n+2m) + 2 = n + 2(m+1)$$

Exercises 7.5

1. Covert the following definiton of relation $F \subseteq \mathbb{N}^2 \times \mathbb{N}$ into functional form:

1.	for all y, $((0,y),0) \in F$
2.	$((x,y),k) \in F \Rightarrow ((x+1,y),k+y) \in F$
3.	nothing else

2. Covert the following definiton of relation $F \subseteq \mathbb{N}^2 \times \mathbb{N}$ into functional form:

1.	for all x, $((x,0),x) \in G$
2a.	for all y, $((0,y),0) \in G$
2b.	$((x,y),k) \in F \Rightarrow ((x+1,y+1,k) \in G$
3.	nothing else

3. For F and G defined above, compute in relational form

(a) $F(2,3)$

(b) $G(5,2)$

(c) $G(2,4)$

4. Repeat Exercise 3 using the functional forms from Exercises 1 and 2.

5. Consider the following functional definition for $F\colon \mathbb{N} \to \mathbb{N}$:

1.	if $x > 100$, then $F(x) = x - 10$
2.	if $x \leq 100$, then $F(x) = F(F(x+11))$

Compute $F(97)$ using functional form.

6. Consider the set $\mathit{Num} \subseteq \{\mathtt{0},\mathtt{1}\}^+$ defined as follows:

1.	$\mathtt{1} \in \mathit{Num}$
2a.	if $x \in \mathit{Num}$, then $x\mathtt{0} \in \mathit{Num}$
2b.	if $x \in \mathit{Num}$, then $x\mathtt{1} \in \mathit{Num}$
3.	nothing else

Num is clearly the set of binary numerals with a leading `1`. In either functional or relational form, define a function $B\colon \mathit{Num} \to \mathbb{N}$ that sends each numeral to the integer it represents.

7. In your favorite programming language, write a program that reads a sequence of `0`s and `1`s and translates the sequence into an integer.

8. Let $F \subseteq \mathbb{N} \times \mathbb{N}$ be defined as follows:

$$\begin{array}{ll} 1. & (0,2) \in F \\ 2. & (x,y) \in F \Rightarrow (x+1, y+3) \in F \\ 3. & \text{nothing else} \end{array}$$

Prove that $f = \{(x, 3x+2) \mid x \in \mathbb{N}\}$.

9. (Hard) Consider the function F defined in Exercise 5. Prove that if $x \leq 100$, then $F(x) = 91$. (*Hint:* Translate into relational form and use induction on derivation sequences with the indcution hypothesis: "If (x,y) has a derivation with length $j \leq k$, then ....")

10. Define the *lexicographic ordering* of $\mathbb{N} \times \mathbb{N}$, denoted by '$\preceq$', as follows: $(n,m) \preceq (k,\ell)$ iff $(n < k)$ or both $(n \leq k)$ and $(m \leq \ell)$). A *Principle of Lexicographic Induction* for predicate H on $\mathbb{N}^2$ is:

If you can prove that

BASE CASE *$P(0,0)$ holds.*

INDUCTION CASE *If $H(i,j)$ holds for all $(j,j) \preceq (k,\ell)$ then $H(k,\ell)$ also holds.*

Then you can conclude that H holds for all for all $(n,m) \in \mathbb{N}^2$

Prove that this induction principle is valid.

11. At the end of this section the function $F\colon \mathbb{N}^2 \to \mathbb{N}$ was recursively defined according to:

$$\begin{array}{rl} 1. & F(0,0) = 0 \\ 2a. & F(n+1,m) = F(n,m) + 1 \\ 2b. & F(n,m+1) = F(n,m) + 2 \end{array}$$

It was proved by structural induction on F (considered as a set) that for all pairs (n,m) *in the domain of* F, $F(n,m) = n + 2m$. Use lexicographic induction (see Exercise 10 to prove that for all $(n,m) \in \mathbb{N}^2$, $F(n,m) = n + 2m$.

7.6 Reasoning about Recursive Functions

The primary method for reasoning about recursive functions is structural induction. We look at a number of examples dealing with the words of the following simple language.

For alphabet $V = \{\texttt{a}, \texttt{b}, \bullet\}$, define the language $L \subseteq V^+$ inductively, according to

$$\begin{array}{rl} 1. & \bullet \in L \\ 2a. & u \in L \Rightarrow \texttt{a}u \in L \\ 2b. & u \in L \Rightarrow \texttt{b}u \in L \\ 3. & \text{nothing else} \end{array}$$

L consists of words over $\{\texttt{a}, \texttt{b}\}$ to which the symbol '$\bullet$' has been appended on the right:

$$L = \{\bullet, \texttt{a}\bullet, \texttt{b}\bullet, \texttt{aa}\bullet, \texttt{ab}\bullet, \texttt{ba}\bullet, \texttt{bb}\bullet, \texttt{aaa}\bullet, \ldots\}$$

Let P be any predicate on L. Since it has a single-element base set and two constructor functions, a proof by structural induction on L has the following form.

Theorem. For all $w \in L$, $H(w)$.

PROOF: The proof is by induction on $u \in L$ with hypothesis $H(u)$

BASE CASE: *A direct proof of* $P(\bullet)$.

INDUCTION STEP: *Proofs that*

$$P(u) \Rightarrow P(\texttt{a}u)$$

and

$$P(u) \Rightarrow P(\texttt{b}u)$$

Often, the two arguments are so similar that only one is shown.

□

We shall define three functions on the language L.

Invert The function $I\colon L \to L$ changes all '$\texttt{a}$'s in a word to '$\texttt{b}$'s and all '$\texttt{b}$'s to '$\texttt{a}$'s. I is defined recursively according to

$$\begin{array}{rl} 1. & I(\bullet) = \bullet \\ 2a. & I(\texttt{a}u) = \texttt{b}I(u) \\ 2b. & I(\texttt{b}u) = \texttt{a}I(u) \end{array}$$

For example $I(\texttt{baab}\bullet) = \texttt{abba}\bullet$ because:

$$\begin{array}{ll} I(\texttt{baab}\bullet) & \\ \quad = \texttt{a}I(\texttt{aab}\bullet) & \text{(I.2b)} \\ \quad = \texttt{ab}I(\texttt{ab}\bullet) & \text{(I.2a)} \\ \quad = \texttt{abb}I(\texttt{b}\bullet) & \text{(I.2a)} \\ \quad = \texttt{abba}I(\bullet) & \text{(I.2b)} \\ \quad = \texttt{abba}\bullet & \text{(I.1)} \end{array}$$

Append The function $A\colon L \times L \to L$ joins two words together. A is defined recursively according to

$$\begin{array}{rl} 1. & A(\bullet, u) = u \\ 2a. & A(\texttt{a}u, v) = \texttt{a}A(u, v) \\ 2b. & A(\texttt{b}u, v) = \texttt{b}A(u, v) \end{array}$$

A is like a concatenation operation, but it throws away the '$\bullet$' symbol between the words. For example,

$$A(\texttt{aa}\bullet, \texttt{bba}\bullet) \overset{2a}{=} \texttt{a}A(\texttt{a}\bullet, \texttt{bba}\bullet) \overset{2a}{=} \texttt{aa}A(\bullet, \texttt{bba}\bullet) \overset{1}{=} \texttt{aabba}\bullet$$

Reverse The function $R\colon L \to L$ reverses the order of '$\texttt{a}$'s and '$\texttt{b}$'s in a word. Reverse uses Append. R is defined recursively according to

$$\begin{array}{rl} 1. & R(\bullet) = \bullet \\ 2a. & R(\texttt{a}u) = A(R(u), \texttt{a}\bullet) \\ 2b. & R(\texttt{b}u) = A(R(u), \texttt{b}\bullet) \end{array}$$

We shall now prove a series of facts about the three functions just defined.

Proposition 7.4 *For all $u \in L$, $I(I(u)) = u$.*

Proof: The proof is by induction on $u \in L$ In the base case, by the defintion of I, rule 1,

$$I(I(\bullet)) = I(\bullet) = \bullet$$

For the induction step,

$$\begin{aligned} I(I(\texttt{a}u)) &= I(\texttt{b}I(u)) && \text{(I.2a)} \\ &= \texttt{a}I(I(u)) && \text{(I.2b)} \\ &= \texttt{a}u && \text{(I.H.)} \end{aligned}$$

Similarly,

$$I(I(\texttt{b}u)) \overset{2b}{=} I(\texttt{a}I(u)) \overset{2a}{=} \texttt{b}I(I(u)) \overset{IH}{=} \texttt{b}u$$

□

From now on, the second proof of the induction case will not be shown unless it differs significantly from the first.

Proposition 7.5 *I distributes over A, that is, for all $u, v \in L$, $I(A(u, v)) = A(I(u), I(v))$.*

PROOF: The proof is by induction on $u \in L$. In the base case,

$$I(A(\bullet, v)) \overset{A.1}{=} I(v) \overset{A.1}{=} A(\bullet, I(v)) \overset{I.1}{=} A(I(\bullet), I(v))$$

For the induction step,

$$\begin{aligned} I(A(\mathtt{a}u, v)) &= I(\mathtt{a}A(u, v)) && \text{(A.2a)} \\ &= \mathtt{b}I(A(u, v)) && \text{(I.2a)} \\ &= \mathtt{b}A(I(u), I(v)) && \text{(I. H.)} \\ &= A(\mathtt{b}I(u), I(v)) && \text{(A.2b)} \\ &= A(I(\mathtt{b}u), I(v)) && \text{(I.2b)} \end{aligned}$$

The proof that $I(A(\mathtt{b}u, v)) = A(I(\mathtt{b}u), I(v))$ is similar.

□

In this proposition there is a choice of two variables on which to perform the induction. The phrase "by induction on u," signals the choice. The induction hypothesis becomes,

$$H(u) \equiv \textit{for all } v \in L,\ I(A(u, v)) = A(I(u), I(v))$$

That is, the for-all quantifier on v is retrained. In this case, u is the appropriate choice because only the first argument varies in the definiiton of A. In the next proposition, we have three variables to choose from.

Proposition 7.6 *A is associative, that is, for all $u, v,$* and w in L

$$A(u, A(v, w)) = A(A(u, v), w)$$

PROOF: The proof is by induction on $u \in L$.
BASE CASE:
$$A(\bullet, A(v, w)) \overset{A.1}{=} A(v, w) \overset{A.1}{=} A(A(\bullet, v), w)$$

INDUCTION STEP:
$$\begin{aligned} A(\mathtt{a}u, A(v, w)) &= \mathtt{a}A(u, A(v, w)) && \text{(A.2a)} \\ &= \mathtt{a}A(A(u, v), w) && \text{(I. H.)} \\ &= A(\mathtt{a}A(u, v), w) && \text{(A.2a)} \\ &= A(A(\mathtt{a}u, v), w) && \text{(A.2a)} \end{aligned}$$

The proof that $A(\mathtt{b}u, A(v, w)) = A(A(\mathtt{b}u, v), w)$ is similar.

□

Proposition 7.7 *$\bullet$ is a right identity for A, that is, for all $u \in L$ $A(u, \bullet) = u$.*

PROOF: Exercise 3.

□

Proposition 7.8 *R and A obey the following distributive law: for all $u, v \in L$,*

$$R(A(u,v)) = A(R(v), R(u))$$

[Note how the positions of u and v are switched.]

PROOF: Exercise 4. □

Proposition 7.9 *R is self cancelling, that is, for all $u \in L$ $R(R(u)) = u$.*

PROOF: The proof is by induction on u.
BASE CASE: by the definition of R,

$$R(R(\bullet)) = R(\bullet) = \bullet$$

INDUCTION STEP:

$$\begin{aligned}
R(R(\mathtt{a}u)) &= R(A(R(u), \mathtt{a}\bullet)) && \text{(A.2a)} \\
&= A(R(\mathtt{a}\bullet), R(R(u))) && \text{(Proposition 7.8)} \\
&= A(R(\mathtt{a}\bullet), u) && \text{(I. H.)} \\
&= A(A(R(\bullet), \mathtt{a}\bullet), u) && \text{(R(2a)} \\
&= A(A(\bullet, \mathtt{a}\bullet), u) && \text{(R.1)} \\
&= A(\mathtt{a}\bullet, u) && \text{(A.1)} \\
&= \mathtt{a}A(\bullet, u) && \text{(A.2a)} \\
&= \mathtt{a}u && \text{(A.1)}
\end{aligned}$$

The proof that $R(R(\mathtt{b}u)) = \mathtt{b}u$ is similar. □

Perhaps you have noticed that in each of the proofs of this section, the necessary facts had conveniently been established by previous propositions. Proposition 7.9 uses the result of Proposition 7.8, which in turn needs the results of Propositions 7.6 and 7.7. The process of *conceiving* a proof typically works in the other direction. It is a goal directed activity just like programming.

In attempting to prove Proposition 7.9, one gets stuck at the second line of the induction step, discovering there that some kind of distributive law is needed. Notice, however, that the distributive law proved in Proposition 7.8 is more general than what is needed for Proposition 7.9. It is usually a good idea to prove a more general fact if you can; most cases it is easier, and the result you have proven may also be applicable to other problems.

Let us define a new function, $T\colon L^2 \to L$, according to

$$\begin{array}{rl}
1. & T(\bullet, v) = v \\
2a. & T(\mathtt{a}u, v) = T(u, \mathtt{a}v) \\
2b. & T(\mathtt{b}u, v) = T(u, \mathtt{b}v)
\end{array}$$

If you evaluate a few examples (Exercise 5) you may discover that T is related to both A and R. In particular, T is equivalent to R when its second argument is $\bullet$:

Proposition 7.10 *For all $u \in L$, $T(u, \bullet) = R(u)$.*

PROOF: Exercise 7. $\square$

The function R might be considered a more natural description of "reversing a word." The function T might be considered a better description of the reversing *computation*. It is easier to prove, for example, that R is self-cancelling and distributes over A than it is to prove these facts about T. On the other hand, there is a definite sense in which T seems more efficient—to see this, evaluate $R(\mathtt{aab}\bullet)$ and $T(\mathtt{aab}\bullet, \bullet)$, according to their definitions, each time counting the number of concatenations performed.

$$\begin{aligned}
&R(\mathtt{aab}\bullet)\\
&= A(R(\mathtt{ab}\bullet), \mathtt{a}\bullet)\\
&= A(A(R(\mathtt{b}\bullet), \mathtt{a}\bullet), \mathtt{a}\bullet)\\
&= A(A(A(\bullet, \mathtt{b}\bullet), \mathtt{a}\bullet), \mathtt{a}\bullet)\\
&= A(A(\mathtt{b}\bullet, \mathtt{a}\bullet), \mathtt{a}\bullet)\\
&= A(\mathtt{b}A(\bullet, \mathtt{a}\bullet), \mathtt{a}\bullet)\\
&= A(\mathtt{ba}\bullet, \mathtt{a}\bullet)\\
&= \mathtt{b}A(\mathtt{a}\bullet, \mathtt{a}\bullet)\\
&= \mathtt{ba}A(\bullet, \mathtt{a}\bullet)\\
&= \mathtt{baa}\bullet
\end{aligned}$$

$$\begin{aligned}
&T(\mathtt{aab}\bullet, \bullet)\\
&= T(\mathtt{ab}\bullet, \mathtt{a}\bullet)\\
&= T(\mathtt{b}\bullet, \mathtt{aa}\bullet)\\
&= T(\bullet, \mathtt{baa}\bullet)\\
&= \mathtt{baa}\bullet
\end{aligned}$$

Exercises 7.6

1. Use the definitions of A and R to compute the values of

(a) $R(\mathtt{abb}\bullet)$

(b) $R(A(\mathtt{a}\bullet, \mathtt{b}\bullet))$

2. Use the definitions of A, R, and T to compute the values of

(a) $R(\mathtt{aab}\bullet)$

(b) $T(\mathtt{aab}\bullet, \bullet)$

(c) $A(R(\mathtt{a}\bullet), \mathtt{b}\bullet)$

(d) $T(\mathtt{a}\bullet, \mathtt{b}\bullet)$

3. Prove Proposition 7.7.

4. Prove Proposition 7.8.

5. Evaluate the following, using the definitions of I, A, R, and T in this section.

(a)	$I(\texttt{abbab}\bullet)$	(b)	$I(I(\texttt{abb}\bullet))$
(c)	$A(\texttt{aa}\bullet, \texttt{ba}\bullet)$	(d)	$A(I(\texttt{ab}\bullet), I(\bullet))$
(e)	$R(\texttt{abb}\bullet)$	(f)	$R(I(\texttt{ab}\bullet))$
(g)	$T(\texttt{baa}\bullet, \texttt{ab}\bullet)$	(h)	$T(\texttt{babb}\bullet, \bullet)$

6. Without referring to the proof in text book, try to prove Proposition 7.8, developing auxiliary propositions as the need arises.

7. Prove Proposition 7.10. [*Hint: You will need to prove a more general fact.*]

8. For the following problems, consider the language $N \subseteq \{\texttt{S}, \bullet\}$, defined inductively as follows:

1. $\bullet \in N$
2. $u \in N$ implies $\texttt{S}u \in N$
3. nothing else

Define the function $P\colon N^2 \to N$ recursively, according to:

1. $P(\bullet, v) = v$
2. $P(\texttt{S}u, v) = \texttt{S}P(u, v)$

Define the function $M\colon N^2 \to N$ recursively, according to:

1. $M(\bullet, v) = \bullet$
2. $M(\texttt{S}u, v) = P(v, M(u, v))$

[*Hint: It may be helpful to express these results using infix notation. It will be helpful to have an idea in mind of what these functions represent.*]

(a) Prove that M distributes over P; that is, for all $u, v, w \in N$, $M(u, P(v, w)) = P(M(u, v), M(u, w))$.

(b) Prove that P is *commutative*; that is, For all $u, v \in N$, $P(u, v) = P(v, u)$.

(c) Prove: For all $u, v \in N$, $M(u, \texttt{S}v) = P(u, M(u, v))$.

(d) Prove: For all $u, v \in N$, $P(\texttt{S}u, v) = P(u, \texttt{S}v)$.

(e) Prove: For all $u \in N$, $P(u, \bullet) = u$.

7.7 Additional Problems

Most of these exercises have to do with applications of recursion and induction to program performance analysis.

Exercises 7.7

1. (hard) Prove:For all $n \in \mathbb{N}$,

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

Now take another look at Exercise 4.3.5. Hint: This problem's proof uses the result of another exercise in this book.

2. Let $A = \{\mathtt{0}, \mathtt{1}\}$ and consider the language of *binary strings*, A^*. A binary string $b_n\, b_{n-1} \cdots b_2\, b_1\, b_0$ Can be interpreted as a *base 2 numeral* representing

$$\sum_{i=0}^{n} \hat{b_i} \cdot 2^i$$

where $\hat{b}$ is the numeric value of "bit" b, namely

$$\hat{b} = \begin{cases} \text{the } \textit{number } 0 \text{ if } b \text{ is the letter } \mathtt{0} \\ \text{the } \textit{number } 1 \text{ if } b \text{ is the letter } \mathtt{1} \end{cases}$$

For alphabet $B = \{\mathtt{0}, \mathtt{1}, \bullet\}$, define the language $L \subseteq B^+$—similar to similar the L in Section 7.6:

$$\begin{array}{rl} 1. & \bullet \in L \\ 2a. & u \in L \Rightarrow \mathtt{0}\hat{\ }u \in L \\ 2b. & u \in L \Rightarrow \mathtt{1}\hat{\ }u \in L \\ 3. & \text{n. e.} \end{array}$$

(a) Define a recursive function $V : L \to \mathbb{N}$ that gives the binary value of a word in L.

(b) Define a recursive function $I : L \to L$ that, given the binary word representing the number $n \in \mathbb{N}$, returns the word representing $n+1$. That is, prove that

$$\text{For all } w \in L,\ V(I(w)) = V(w) + 1$$

(c) Define an "addition" function, $A : (L \times L) \to L$, and prove

$$\text{For all } u, v \in L,\ V(A(u, v)) = V(u) + V(v)$$

3. Define $F : \mathbb{N} \to \mathbb{N}$ and $G : \mathbb{N}^2 \to \mathbb{N}$ as follows:

$$\begin{array}{rcl} F(0) & = & 1 \\ F(k+1) & = & (k+1) \cdot F(k) \end{array} \qquad \begin{array}{rcl} G(0, m) & = & m \\ G(k+1, m) & = & G(k,\ m \cdot (k+1)) \end{array}$$

(a) Prove by induction on $n \in \mathbb{N}$: For all $n, m \in \mathbb{N}$, $G(n, m) = m \cdot G(n, 1)$.

(b) Prove: For all $n \in \mathbb{N}$, $F(n) = G(n, 1)$.

4. Define $F : \mathbb{N} \to \mathbb{N}$ and $FT : \mathbb{N}^3 \to \mathbb{N}$ as follows:

$$\begin{array}{rcl} F(0) & = & 1 \\ F(1) & = & 1 \\ F(k+2) & = & F(k) + F(k+1) \end{array} \qquad \begin{array}{rcl} FT(0, n, m) & = & n \\ FT(k+1, n, m) & = & FT(k, m, n+m)) \end{array}$$

Prove: For all $n \in \mathbb{N}$, $F(n) = FT(n, 1, 1)$

5. Let $a \in \mathbb{N}$. The function $T : \mathbb{N} \to \mathbb{N}$ is defined recursively by

$$\begin{array}{rcl} T(0) & = & 0 \\ T(1) & = & a + \frac{1}{2} \\ \\ T(k+2) & = & 2T(k+1) - T(k) + 1 \end{array}$$

Prove that $(n^2/2) + an$ satisfies the recurrence, that is, *for all natural numbers n*

$$T(n) = \frac{n^2}{2} + an$$

6. Let $a \in \mathbb{N}$. The function $T : \mathbb{N} \to \mathbb{N}$ is defined recursively by

$$\begin{array}{rcl} T(0) & = & a \\ T(k+1) & = & T(k) + k + 1 \end{array}$$

Prove that for all $n \in \mathbb{N}$,

$$T(n) = a + \frac{n^2 + n}{2}$$

7. The function $q : \mathbb{N}^2 \to \mathbb{N}$ is defined:

$$\begin{array}{rcl} q(0, m) & = & m \\ q(k+1, m) & = & 1 + q(k, m) \end{array}$$

Prove that for all $n \in \mathbb{N}$, $q(n, 0) = q(0, n)$.

7.8 Validity of the Induction Principle*

The goal in this section is to establish the truth of Theorem 7.3; that is, to establish rigorously and in general that inductive arguments like those of the previous section are valid. We shall do this by showing how structural induction is simply an encoded form of mathematical induction. Once we have proved the basic principle, we can safely go on to reason at a higher level.

We characterized the elements of the language L and the relation R as having "derivations" from base elements. The next definition formalizes the notion of derivation as a *construction sequence*. The definition accounts for the general case in which there are several constructor functions.

Definition 7.4 *Let U be a set, B a subset of U, and f_1, f_2, ..., f_m a collection of functions on U of various ranks. An element $a \in U$ is said to have a* construction sequence in U from B under f_1, f_2, ..., f_m *if there exists a sequence of elements in U,*

$$\langle u_1, u_2, \ldots, u_n\rangle,\ n \geq 1$$

with the property that each u_i is either:

(a) *an element of B, or*

(b) *$u_i = f_j(a_1, \ldots, a_r)$, where f_j is an $r-$place function and each argument a_k occurs prior to u_i in the sequence.*

Example 7.10 In the language L defined in Example 7.1, we had the following construction sequence for the word `width*height*5`:

$$\langle \texttt{width},\ \texttt{height},\ \texttt{width*height},\ \texttt{5},\ \texttt{width*height*5}\rangle$$

There are many other possible constructions sequences for the same word, for instance,

$$\langle \texttt{5},\ \texttt{height},\ \texttt{height*5},\ \texttt{5},\ \texttt{width},\ \texttt{5*width},\ \texttt{5},\ \texttt{width*height*5}\rangle$$

Check that both of these sequences are built according to rules (a) and (b) of Definition 7.4; so they are constructions sequences for `width*height*5`. □

Example 7.11 In the relation R of Example 7.2, the pair $(5, 10)$ has a construction sequence

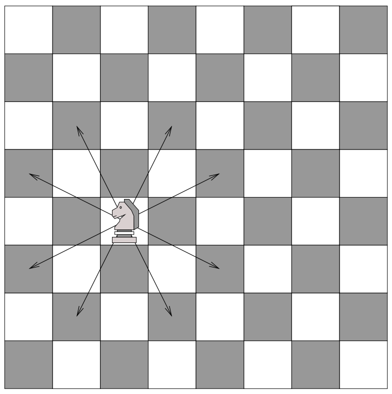
$$\langle(0,0),\ (1,2),\ (2,4),\ (3,6),\ (4,8),\ (5,10)\rangle$$

There are many possible construction sequences for $(5, 10)$ but this is the shortest one. □

Example 7.12 Let $B = \{1, 2,\ \ldots,\ 8\}$ and let $U = B^2$. Define the following eight functions on U:

$f_1(r,c) = (r+2, c+1)$ if $r \leq 6$ and $c \leq 7$; (r,c) otherwise.
$f_2(r,c) = (r+1, c+2)$ if $r \leq 7$ and $c \leq 6$; (r,c) otherwise.
$f_3(r,c) = (r-1, c+2)$ if $r \geq 1$ and $c \leq 6$; (r,c) otherwise.
$f_4(r,c) = (r-2, c+1)$ if $r \geq 2$ and $c \leq 7$; (r,c) otherwise.
$f_5(r,c) = (r-2, c-1)$ if $r \geq 2$ and $c \geq 1$; (r,c) otherwise.
$f_6(r,c) = (r-1, c-2)$ if $r \geq 1$ and $c \geq 2$; (r,c) otherwise.
$f_7(r,c) = (r+1, c-2)$ if $r \leq 7$ and $c \geq 2$; (r,c) otherwise.
$f_8(r,c) = (r+2, c-1)$ if $r \leq 6$ and $c \geq 1$; (r,c) otherwise.

Think of U as a chess board. These functions represent all the possible moves of a knight:



Now pick a square on the board, say $(3, 4)$. The construction sequences in U from $\{(3, 4)\}$ under $f_1, \ldots, f_8$ generate all the squares that a knight can reach starting from there.

$\square$

Theorem 7.11 (Fundamental Theorem on Induction) *The set A of all elements of U that have a construction sequence from B under constructors f_1, $\ldots$, f_m is inductively defined from B and f_1, $\ldots$, f_m. That is, A is the smallest set containing B and closed under each of the constructors.*

PROOF: The proof has three parts. We must show that A contains B, that A is closed, and finally, that A is the smallest such set.
CLAIM I $B \subseteq A$
If $b \in B$ then by Definition 7.4, $\langle b \rangle$ is a construction sequence. Hence, $b \in A$ and so we have shown that $B \subseteq A$. This proves Claim I.
CLAIM II A is closed with respect to each constructor.
Suppose that $a_1, \ldots, a_r \in A$. Then each a_i has a construction sequence, $\langle u_{i1}, \ldots, a_i \rangle$. But then we can get a construction sequence for $f(a_1, \ldots, a_r)$ by putting all these sequences together end-to-end in any order:

$$\langle u_{11}, \ldots, a_1, u_{21}, \ldots, a_2, \ldots, u_{r1}, \ldots, a_r, f(a_1, \ldots, a_r) \rangle$$

Therefore, $f(a_1, \ldots, a_r) \in A$ and we have shown that A is closed with respect to f. Since f was arbitrary, A is closed with respect to all the constructors. This proves Claim II.
CLAIM III If $S \subseteq U$ contains B and is closed with respect to each f_1, $\ldots$, f_n, then $A \subseteq S$.

To prove the claim, let S be as assumed. We want to show that $A \subseteq S$. By Definition 7.2, it suffices to show that any every element with a construction sequence must be in S. This is proven by induction on $k \in \mathbb{N}$ with hypothesis

$H(k) \equiv$ *Every element with a construction sequence of length k or less is in S*

BASE CASE: The base case holds vaccuously because there are no construction sequences of length 0.
INDUCTION STEP: Assume $H(k)$ and suppose that u has a construction sequence

$$d = \langle v_1, v_2, \ldots, v_k, u \rangle$$

By Definition 7.4, u is either an element of B or $u = f(a_1, \ldots, a_r)$ with each a_i occurring in d. In the first case, since $B \subseteq S$, we know that $u \in S$. In the second case we can, without loss of generality, assume that

$$d = \langle\ \ldots, a_1, \ldots, a_2, \ldots, a_r, \ldots, u \rangle$$

Then each a_i has a construction sequence,

$$d_i = \langle\ \ldots, a_1, \ldots, a_2, \ldots, a_i \rangle$$

By the induction hypothesis, this implies that each $a_i \in S$, and since S is closed with respect to f, $f(a_1, \ldots, a_r) = u \in S$. This concludes the induction step, the proof of Claim III, and the proof of the theorem.

□

Theorem 7.11 shows us one way to construct inductively defined sets—and hence that such sets exist. Proposition 7.1 confirms that they are uniquely defined. Because of these results, we know that we can specify a set exactly, just by describing its base set and constructor function(s). We do not have to mention construction sequences, although it is sometimes useful to remember that they exist (for example, see Exercise 5 in Section 7.5).

As the exercises at the end of this section suggest, our definition of inductive sets could be more general than it is. Of particular importance is the idea of simultaneously defining several sets inductively (see Exercise 5). We will see this kind of construction many times in later chapters. The methods for determining the uniqueness of these sets is the same: they are determined by looking at their constructions sequences, or equivalently, by specifying their closure properties. It is hard to come up with a *most* general set definition scheme. On the other hand, if a novel scheme is needed it is straightforward to prove a version of the Fundamental Theorem for it.

Construction sequences are one mathematical mechanism for building inductively defined sets, but there are others. Exercise 3 illustrates another way that is commonly seen, and asks you to prove that the two constructions are equivalent.

Our goal is to find a way to characterize inductively defined sets that is independent of the "mechanics" of how they are built. The notation scheme of Definition 7.3 gives a mechanim-independent way to define sets. The Principle of Structural Induction gives a mechanism-independent way to ask questions about them. We now prove the validity of the Principle.

Theorem 7.3 (restated) Let $A \subseteq U$ be inductively defined from base set B and functions $f_1, \ldots, f_m$. If $P\colon U \to \{T, F\}$ has the following properties:

(BASE CASE) For all $x \in B$, $P(x)$ holds.

(INDUCTION STEP) For each r-place constructor f,

$$P(x_1) \wedge \cdots \wedge P(x_r) \quad \Rightarrow \quad P(f(x_1, \ldots, x_r))$$

Then $P(x)$ holds for all $x \in A$,

PROOF: A is just the set of elements with construction sequences, so the proof is by induction on $k \in \mathbb{N}$ with hypothesis

$H(k) \equiv P$ *holds for every* $x \in U$ *with a construction sequence of length* k *or less.*

BASE CASE: $H(0)$ is true vaccuously.
INDUCTION STEP: Assume $H(k)$ and consider an element x with construction sequence $\langle a_1, a_2, \ldots, a_k, x \rangle$. According to Definition 7.4, either $x \in B$, in which case $P(x) = T$ by assumption (a), or $x = f_i(x_1, \ldots, x_r)$ and each x_i appears earlier in this construction sequence. But if x_i appears in the construction sequence, then x_i has a construction sequence of length k or less, so by the induction hypothesis, $P(a_i) = T$, for $1 \leq i \leq r$. Hence, by assumption (b), $P(x) = T$. This concludes the induction case and the proof of the theorem.

□

Just as Theorem 7.11 allows us to define inductive sets without specifically saying how they might be built, Theorem 7.3 allows us to deduce properties about inductive sets without mentioning construction sequences. The theorem makes the underlying induction argument, once and for all.

Example 7.13 Recall that in Proposition 7.2 we were to show the relation R, defined inductively from base set $\{(, 0, 0)\}$ and function

$$g(n, m) = (n + 1, m + 2),$$

is a subset of $E = \{(x, 2x) \mid x \in \mathbb{N}\}$. □

Here is the proof, expressed as a structural induction.

Claim. For E and R as defined in Proposition 7.2, $R \subseteq E$.

PROOF: The proof is by structural induction on $x \in R$ with hypothesis

$$H(x) \equiv x \in E$$

BASE CASE: $(0, 0) = (0, 2 \cdot 0) \in E$

INDUCTION STEP: Suppose $(n, m) \in E$. Then $(n, m) = (x, 2x)$ for some number x. Then

$$g(n, m) = (n + 1, m + 2) = (x + 1, 2x + 2) = (x + 1, 2(x + 1)) \in E$$

□

Example 7.14 To see what work is saved by the Principle of Structural Induction, let us "translate" the preceding argument into its underlying mathematical induction.

□

PROOF: We prove $R \subseteq E$ by induction on $k \in \mathbb{N}$ with hypothesis,

$$H(k) \equiv \text{if } (n,m) \text{ has a construction of length } k \text{ or less, } (n,m) \in E.$$

BASE CASE: The only construction sequence of length one is $\langle(0,0)\rangle$ and $(0,0) \in E$ as has already been explained.
INDUCTION STEP: Assume $H(k)$, and suppose that (n,m) has a construction sequence of length $k+1$,

$$\langle(r_1,s_1),(r_2,s_2),\ldots,(r_k,s_k),(n,m)\rangle$$

According to Definition 7.4, either $(n,m) = (0,0)$, in which case we already know that $(n,m) \in E$, or $(n,m) = g(r_i,s_i) = (r_i+1, s_i+2)$ for some $i \le k$. By the induction hypothesis, $(r_i,s_i) \in E$, so $s_i = 2r_i$. But then

$$(n,m) = (r_i+1, 2r_i+2) = (r_i+1, 2(r_i+1)) \in E$$

This concludes the induction.

Since R is just the set of all pairs with construction sequences, it follows that $R \subseteq E$.

□

Exercises 7.8

1. If the word "function" is replaced by the phrase "partial function" in Definition 7.4, does it change the validity of Theorem 7.11?

2. If the word "function" is replaced by the word "relation" in Definition 7.4, does it change the validity of Theorem 7.11?

3. Let $f\colon U \to V$ and $A \subseteq U$. Recall (Definition 2.4 that the *image* of A under f is defined to be

$$fA = \{f(x) \mid x \in A\}$$

The following is an alternative definition for *inductively defined set*, for a single constructor function (it can be generalized to many constructors).

> **Definition**. *Let U be a set, $B \subseteq U$, and $f\colon U \to U$. The set A is said to be* inductively defined by stages from B and f *if $A = \bigcup_{k\in\mathbb{N}} A_k$, where*
>
> $$\begin{array}{rcl} A_0 & = & B \\ A_1 & = & A_0 \cup fA_0 \\ & \vdots & \\ A_{k+1} & = & A_k \cup fA_k \\ & \vdots & \end{array}$$

Prove: A is inductively defined from B and f if and only if A is inductively defined by stages from B and f.

4. Given a relation $R \subseteq A \times A$, define the set $R^\star = \bigcup_{k \in \mathbb{N}} R_k$, where

$$\begin{array}{rcl} R_0 & = & R \\ & \vdots & \\ R_{k+1} & = & R_k \cup \{(x,z) \mid \exists y \in A \colon (x,y) \in R_k \text{ and } (y,z) \in R_k\} \\ & \vdots & \end{array}$$

Prove that $R^\star$ is the smallest transitive relation that contains R. $R^\star$ is called the *transitive closure* of R.

5. Let U_1 and U_2 be sets, let $B_1 \subseteq U_1$ and $B_2 \subseteq U_2$; and let $f_1 \colon U_1 \times U - 2 \to U_1$ and $f_2 \colon U_1 \times U_2 \to U_2$. Let

A_1 be the set of all elements of U_1 which have a construction sequence from $B_1 \cup B_1$ under f_1 and f_2

A_2 be the set of all elements of U_2 which have a construction sequence from $B_1 \cup B_1$ under f_1 and f_2

A_1 and A_2 are said to be *simultaneously inductively defined* Prove that A_1 and A_2 are the smallest of all sets S and T such that:

(a) $B_1 \subseteq S$ and $B_2 \subseteq T$

(b) For all $x \in S$ and $y \in T$, $f_1(x,y) \in S$ and $f_2(x,y) \in T$.

6. (tedious) Generalize the previous exercise to an arbitrary number of simultaneously defined sets under an arbitrary number of constructor functions (or relations).

Chapter 8

Languages and Meanings

In this chapter we look at how programming languages are defined. There are two aspects to consider. We must develop a method to specify exactly what words are valid as program expressions. And we must develop a method to say precisely what computation a valid program expresses. We now have the basic mathematical tools to make these specifications: programming languages are specified by inductive set definitions and their meanings by recursive function definitions. However, we still must develop techniques to use these tools effectively.

8.1 Language Definitions

We shall start with a seemingly simple language of expressions. The idea is to put the language together by showing how to build more complicated expressions from simpler ones. This is the way almost all computer languages are defined.

Let the set $A = \mathbb{N} \cup \{\,\$, \#\,\}$ be our alphabet of symbols. Define the language $L \subseteq A^+$ inductively, according to

1.	$\mathbb{N} \subseteq L$
2a.	$u, v \in L \Rightarrow u \ \$ \ v \in L$
2b.	$u, v \in L \Rightarrow u \ \# \ v \in L$
3.	nothing else

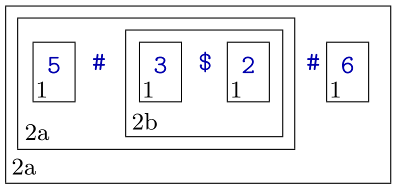
The first kind of question that might be asked is whether a particular word in A^+ is in (i.e. an element of) the language L. To answer such a question, we must analyze the given word to see whether it could be built using the rules of L's definition.

Example 8.1 Is the word $5 \ \# \ 3 \ \$ \ 2 \ \# \ 6$ in L? $\square$

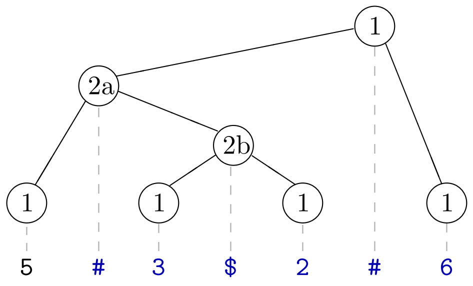
The answer is "yes," because there is a construction sequence:

1.	$3 \in L$	by rule 1
2.	$2 \in L$	by rule 1
3.	3 \$ $2 \in L$	by rule 2b, 1 and 2
4.	$5 \in L$	by rule 1
5.	5 # 3 \$ $2 \in L$	by rule 2a, 4 and 3
6.	$6 \in L$	by rule 1
7.	5 # 3 \$ 2 # $6 \in L$	by rule 2a, 5 and 6

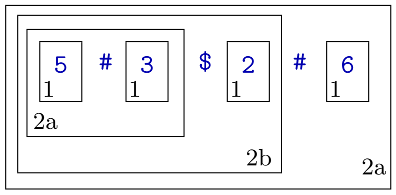
The analysis showing that a word is in a language is called *parsing*. There are two ways to diagram derivations like this. A *parsing diagram* uses nested boxes to show how the word decomposes:



The second way is to draw a A *parse tree*, whose interior nodes are labeled by rules and whose leaves are symbols of the alphabet:



Each of the diagrams above reflects the same parse. Here is a different parse showing that 5 # 3 \$ 2 # 6 is in L:



That there are several ways to prove that a given word is in L is a problem, as we shall see next.

8.2 Defining How Languages are Interpreted

The interpretation of a language associates a value with each word. The word is said to express the value. If the language is inductively defined, then the interpretation can be defined recursively.

Using the language L of the previous section, let us define a function $\mathcal{V}\colon L \to \mathbb{N}$, which gives a natural-number interpretation where symbols # and $ express addition and multiplication, respectively.

1.	for $k \in \mathbb{N}$,	$\mathcal{V}[k]$	$= k$
2a.	for words of the form $w = u \mathbin{\#} v$,	$\mathcal{V}[w]$	$= \mathcal{V}[u] + \mathcal{V}[v]$
2b.	for words of the form $w = u \mathbin{\$} v$,	$\mathcal{V}[w]$	$= \mathcal{V}[u] \times \mathcal{V}[v]$

Based on this definition and the first parsing analysis, we can determine an interpretation of 5 # 3 $ 2 # 6:

$$\begin{array}{lll}
1. & 3 \in \mathbb{N} & \Rightarrow \mathcal{V}[3] = 3 \\
2. & 2 \in \mathbb{N} & \Rightarrow \mathcal{V}[2] = 2 \\
3. & \left.\begin{array}{lcl} u & = & 3 \in L \\ v & = & 2 \in L \end{array}\right\} & \Rightarrow \mathcal{V}[u \mathbin{\$} v] = \mathcal{V}[u] \times \mathcal{V}[v] = 3 \times 2 = 6 \\
4. & 5 \in \mathbb{N} & \Rightarrow \mathcal{V}[5] = 5 \\
5. & \left.\begin{array}{lcl} u & = & 5 \in L \\ v & = & 3 \mathbin{\$} 2 \in L \end{array}\right\} & \Rightarrow \mathcal{V}[u \mathbin{\#} v] = \mathcal{V}[u] + \mathcal{V}[v] = 5 + 6 = 11 \\
6. & 6 \in \mathbb{N} & \Rightarrow \mathcal{V}[6] = 6 \\
7. & \left.\begin{array}{lcl} u & = & 5 \mathbin{\#} 3 \mathbin{\$} 2 \in L \\ v & = & 6 \in L \end{array}\right\} & \Rightarrow \mathcal{V}[u \mathbin{\#} v] = \mathcal{V}[u] + \mathcal{V}[v] = 11 + 6 = 17
\end{array}$$

That is, $\mathcal{V}[5 \mathbin{\#} 3 \mathbin{\$} 2 \mathbin{\#} 6] = 17$. However, a different parse leads to a different interpretation. The derivation that follows is "top-down" in the sense that the interpreted expression is decomposed as the interpretation is applied. At each step, you should verify that the word is broken down in a manner that is consistent with L's definition.

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathcal{V}[5 \mathbin{\#} 3 \mathbin{\$} 2 \mathbin{\#} 6]} \\
= & \mathcal{V}[5] + \mathcal{V}[3 \mathbin{\$} 2 \mathbin{\#} 6] & \text{defn. } \mathcal{V}\text{, case 2a} \\
= & 5 + \mathcal{V}[3 \mathbin{\$} 2 \mathbin{\#} 6] & \mathcal{V}(1) \\
= & 5 + \big(\mathcal{V}[3] \times \mathcal{V}[2 \mathbin{\#} 6]\big) & \mathcal{V}(2b) \\
= & 5 + \big(3 \times \mathcal{V}[2 \mathbin{\#} 6]\big) & \mathcal{V}(1) \\
= & 5 + \big(3 \times \big(\mathcal{V}[2] + \mathcal{V}[6]\big)\big) & \mathcal{V}(2a) \\
= & 5 + \big(3 \times \big(2 + 6\big)\big) & \mathcal{V}(1)\text{, twice} \\
= & 29 & \text{arithmetic}
\end{array}$$

That is, $\mathcal{V}[5 \mathtt{\#} 3 \mathtt{\$} 2 \mathtt{\#} 6] = 17 = 29$ (?!). The apparent contradiction is due to the assumption that $\mathcal{V}$ is a function; use of the '=' symbol, in defining $\mathcal{V}$ and in deriving a value, is simply wrong. Both interpretations are correct: $\mathcal{V}$ is a *relation* associating the values 17, 29, and several others, with the word $5 \mathtt{\#} 3 \mathtt{\$} 2 \mathtt{\#} 6$.

When computer languages are defined, it is usually intended that each word have a unique interpretation—we *want* $\mathcal{V}$ to be a function. When multiple interpretations exist, it is said that the language is *ambiguous*. As we shall see later, ambiguity is not necessarily the fault of the language, but even so, languages can be designed to be unambiguous. In the following two examples, variations of L are defined, by which the problems just encountered are avoided.

Example 8.2 We can remove the ambiguity in L by introducing parenthesis symbols. Take $V = \mathbb{N} \cup \{\mathtt{\#}, \mathtt{\$}, \mathtt{(}, \mathtt{)}\}$. The language L_2 and its interpretation relation $\mathcal{V}_2$ are defined simultaneously below:

	$L_2 \subseteq V^+$	$\mathcal{V}_2 \colon L_2 \to \mathbb{N}$
1.	$\mathbb{N} \subseteq L_2$	$\mathcal{V}_2[k] = k,\ \text{for } k \in \mathbb{N}$
2*a*.	$u, v \in L_2 \Rightarrow \mathtt{(}\, u \,\mathtt{\#}\, v \,\mathtt{)} \in L_2$	$\mathcal{V}_2[\mathtt{(}\, u \,\mathtt{\#}\, v \,\mathtt{)}] = \mathcal{V}_2[u] + \mathcal{V}_2[v]$
2*b*.	$u, v \in L_2 \Rightarrow \mathtt{(}\, u \,\mathtt{\$}\, v \,\mathtt{)} \in L_2$	$\mathcal{V}_2[\mathtt{(}\, u \,\mathtt{\$}\, v \,\mathtt{)}] = \mathcal{V}_2[u] \times \mathcal{V}_2[v]$
3.	nothing else	

In this language,

$$\mathcal{V}_2[\mathtt{(}\mathtt{(}5 \,\mathtt{\#}\, \mathtt{(}3 \,\mathtt{\$}\, 2\mathtt{)}\mathtt{)} \,\mathtt{\#}\, 6\mathtt{)}/] = 17$$

and

$$\mathcal{V}_2[\mathtt{(}5 \,\mathtt{\#}\, \mathtt{(}3 \,\mathtt{\$}\, \mathtt{(}2 \,\mathtt{\#}\, 6\mathtt{)}\mathtt{)}\mathtt{)}] = 29.$$

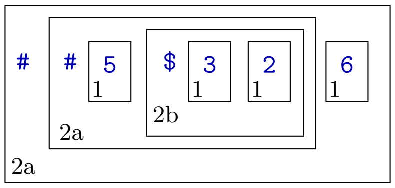
□

This version of L_2 forces every expression to be fully parenthesized, but the interpretation is unambiguous. $\mathcal{V}_2$ is a function because there is only one way to parse any word in L_2.

Example 8.3 In the version of L shown below, there are no parentheses but the operator symbols have been moved from an infix position to a *prefix* position:

	$L_3 \subseteq V^+$	$\mathcal{V}_3 \colon L_3 \to \mathbb{N}$
1.	$\mathbb{N} \subseteq L_3$	$\mathcal{V}_3[k] = k,\ \text{for } k \in \mathbb{N}$
2*a*.	$u, v \in L_3 \Rightarrow \mathtt{\#}\ u\ v \in L_3$	$\mathcal{V}_3[\mathtt{\#}\ u\ v] = \mathcal{V}_3[u] + \mathcal{V}_3[v]$
2*b*.	$u, v \in L_3 \Rightarrow \mathtt{\$}\ u\ v \in L_3$	$\mathcal{V}_3[\mathtt{\$}\ u\ v] = \mathcal{V}_3[u] \times \mathcal{V}_3[v]$
3.	nothing else	

To express 17, one would write:



This is the only way to parse # # 5 $ 3 2 6 because its decomposition under the definition of L_3 is determined by the initial symbol of the word. Only one symbol can be the initial symbol, so there can only be one parse. Consequently, the only possible interpretation is:

$$\begin{array}{lll}
\multicolumn{2}{l}{\mathcal{V}_3[\# \; \# \; 5 \; \$ \; 3 \; 2 \; 6]} & \\
= & \mathcal{V}_3[\# \; 5 \; \$ \; 3 \; 2] + \mathcal{V}_3[6] & \mathcal{V}_3(2a) \\
= & \big([\mathcal{V}_3[5] + \mathcal{V}_3[\$ \; 3 \; 2]\big) + \mathcal{V}_3[6] & \mathcal{V}_3(2a) \\
= & \big([\mathcal{V}_3[5] + \big([\mathcal{V}_3[3] \times \mathcal{V}_3[2]\big)\big) + \mathcal{V}_3[6] & \mathcal{V}_3(2b) \\
= & (5 + (3 \times 2)) + 6 & \mathcal{V}_3[1]\text{, four times} \\
= & 17 & \text{arithmetic}
\end{array}$$

□

Exercises 8.2

1. For the language L and interpretation relation $\mathcal{V}$ defined at the beginning of this section, list all the values that are associated with the expression $\mathcal{V}$[5 # 3 $ 2 # 6].

2. Draw the tree corresponding to the second parse of 5 # 3 $ 2 # 6. with respect to the language L.

3. Using the definition of $\mathcal{V}_3$, check that the expression shown in Example 3 evaluates to 17. Then evaluate the following words of L_3:

(a) # $ 3 # 8 9 $ 2 5

(b) # # # $ 3 4 5 6 7

(c) # 3 # 4 # 5 $ 6 7

4. Give an expression in the language L_3 of Example 8.3 which evaluates to 29.

5. Consider the following language, $L_4 \subseteq V^+$, for $V = \mathbb{N} \cup \{ \texttt{\#} , \texttt{\$} \}$.

	$L_4 \subseteq V^+$
1.	$\mathbb{N} \subseteq L_4$
2*a*.	$u, v \in L_4 \Rightarrow \texttt{\#}\ u\ v \in L_4$
2*b*.	$u, v \in L_4 \Rightarrow u\ v\ \texttt{\$}\ \in L_4$
3.	nothing else

Define an interpretation function for L_4 under which ‘ # ’ stands for addition and ‘ $ ’ for multiplication.

6. For each of the expressions (a)–(c) in Exercise 3, give the corresponding expression in L_4.

7. Determine whether the following words are in L_4, and if so evaluate them:

(a) # # 2 3 4 $ 4

(b) # 2 3 $ 4 5 $

(c) 2 # 3 # 5 $ 6

(d) 2 # 3 # 5 $ 6

(e) 2 3 # 4 # 5 6 $ 7 $ $

(f) 1 # # 2 3 4 5 6 $ 7 8 $ $ 9 $

8.3 Specifying Precedence

We have seen in Examples 8.2 and 8.3 that one can control the way operations are applied by using disambiguating syntax, like parentheses, or otherwise changing the grammar of the language. One can also deal with the problem mathematically by introducing more structure to the language definition. Let $V = \mathbb{N} \cup \{ \texttt{\$}, \texttt{\#} \}$, as before. First, define a language F and interpretation function $\mathcal{V}_F\colon F \to \mathbb{N}$, involving only the ‘$’ operation symbol:

	$F \subseteq V^+$	$\mathcal{V}_F\colon F \to \mathbb{N}$
1.	$\mathbb{N} \subseteq F$	$\mathcal{V}_F[k] = k,$ for $k \in \mathbb{N}$
2.	$u \in \mathbb{N},\ v \in F \Rightarrow u\ \texttt{\$}\ v \in F$	$\mathcal{V}_F[u\ \texttt{\$}\ v] = u \times \mathcal{V}_F[v]$
3.	nothing else	

If you study these definitions carefully, you will see a subtle difference from the earlier definition of L (aside from the fact that part 2a is missing!). Part 2 of the definition, builds and interprets words in such a way that multiplications are carried out from right to left. It is said that ‘$’ *associates to the right.* To illustrate why this must be, let us consider the word $w = 3 \$ 4 \$ 5 \$ 6$. There is exactly one way to parse w:

$$\boxed{\boxed{3}\ \$\ \boxed{\boxed{4}\ \$\ \boxed{\boxed{5}\ \$\ \boxed{6}}}}$$

Hence, there is exactly one interpretation:

$$
\begin{array}{lll}
\mathcal{V}_F[3 \$ 4 \$ 5 \$ 6] & & \\
\quad = & 3 \times \mathcal{V}_F[4 \$ 5 \$ 6] & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \mathcal{V}_F[5 \$ 6]\big) & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \big(5 \times \mathcal{V}_F[6]\big)\big) & \mathcal{V}_F(2) \\
\quad = & 3 \times \big(4 \times \big(5 \times 6\big)\big) & \mathcal{V}_F(1) \\
\quad = & 360 & \text{arithmetic}
\end{array}
$$

In this case, the order of evaluating multiplications doesn't matter; but it *would* matter if ‘$\$$’ were to denote, say, subtraction (or *real* computer multiplication with overflow).

Building from the sublanguage F, we can now create a language T by introducing the addition symbol.

	$T \subseteq \big(F \cup \{\$\}\big)^+$	$\mathcal{V}_F : F \to \mathbb{N}$
1.	$F \subseteq T$	$\mathcal{V}_T[u] = \mathcal{V}_F[u]$, for $u \in F$
2.	$u \in F, v \in T \Rightarrow u \# v \in F$	$\mathcal{V}_T[u \# v] = \mathcal{V}_F[u] + \mathcal{V}_T[v]$
3.	nothing else	

The language T is *exactly the same* as the language L that we originally defined. However, the interpretation, V_T, is constrained to perform both additions and multiplications from right to left. In addition, multiplications are performed before additions. It is said that ‘$\$$’ takes *precedence* over ‘#’. Let us check this with the original problem expression.

$$
\begin{array}{lll}
\mathcal{V}_T[5 \# 3 \$ 2 \# 6] & & \\
\quad = & \mathcal{V}_T[5 \# 3 \$ 2] + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(\mathcal{V}_T[5] + \mathcal{V}_T[3 \$ 2]\big) + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(\mathcal{V}_T[5] + \mathcal{V}_T[3 \$ 2]\big) + \mathcal{V}_T[6] & {}_T(1) \\
\quad = & \big(\mathcal{V}_T[5] + \big(3 \times \mathcal{V}_T[2]\big)\big) + \mathcal{V}_T[6] & {}_T(2) \\
\quad = & \big(5 + \big(3 \times 2\big)\big) + 6 & {}_T(1)\text{, three times} \\
\quad = & 17 & \text{arithmetic}
\end{array}
$$

Since this is the only way to evaluate the word, we are correct in regarding $\mathcal{V}_T$ and V_F as functions.

8.4 Environments

The languages we have seen so far have contained only constants and operations. Computer languages also contain program variables; this is what gives them much of their power. In most languages, a program variable designates some computer-memory location which contains the value. The language compiler translates the symbolic variable name into an address, which is used by the computer to retrieve the designated value.

For our purposes, it is all right to think of the computer's memory as a device that maps the symbolic variable directly to a value; we shall not concern ourselves with the hidden translation from identifier to address. Thus, a memory can be modeled as a function from the domain of program variables to the range of interpreted values. We call such a mapping an *environment*.

As a program executes, its environment changes. The value associated with a program variable is altered by assignment statements, the binding of procedure parameters, and so forth. Our language specifications must reflect this fact and this is done by including the environment in the definition of the interpretation.

Example 8.4 Let IDE be a set of program variables (such as `x` , `alpha` , `I5` , *etc.*). Let ENV be the set of all environments,

$$\text{ENV} = \{\sigma \mid \sigma\colon \text{IDE} \to \mathbb{N}\}$$

Finally, take V to be the alphabet

$$V = \mathbb{N} \cup \text{IDE} \cup \{\texttt{\#}\}$$

and define a prefix (hence unambiguous) language $L_6 \subseteq V^+$ and interpretation function $\mathcal{V}\colon L_6 \times \text{ENV} \to \mathbb{N}$ according to:

	$L_6 \subseteq V^+$	$\mathcal{V}_6\colon L_6 \times \text{ENV} \to \mathbb{N}$
1*a*.	$\mathbb{N} \subseteq L_6$	$\mathcal{V}_6[k](\sigma) = k,\ \text{for } k \in \mathbb{N}$
1*b*.	$\text{IDE} \subseteq L_6$	$\mathcal{V}_6[v](\sigma) = \sigma(v),\ \text{for } v \in \text{IDE}$
2.	$u, v \in L_6 \Rightarrow \texttt{\#}\ u\ v \in L_6$	$\mathcal{V}_6[\texttt{\#}\ u\ v](\sigma) = \mathcal{V}_6[u](\sigma) + \mathcal{V}_6[v](\sigma)$
3.	nothing else	

We write $\mathcal{V}_6[u](\sigma)$ rather than $\mathcal{V}_6[u,\sigma]$ or $\mathcal{V}(u,\sigma)$ because it is a little clearer. Since the environment variable is always a single letter, we will later drop the surrounding parentheses.

Clause 1(b) is new; it specifies the interpretation of a program variable, $v \in \text{IDE}$, whose value is provided by the environment σ. $\square$

Figure 8.4 defines a language of infix arithmetic expressions involving operations of addition, subtraction, multiplication, and negation. Among these operations, negation has the highest precedence, then multiplication, and addition and subtraction have lower, but equal, precedence. Precedence may be superseded by parentheses. All operations are performed from left to right. This example shows that by building the mathematical structures in the right way, we can be precise about the meaning of "ambiguous" languages.

Example 8.5 If environment $\sigma = \{(\texttt{row}, 5), (\texttt{col}, 8)\}$, then

$$
\begin{aligned}
&\mathcal{V}_E[\texttt{5 * (row + 2) * - col)}]\sigma \\
&= \mathcal{V}_T[\texttt{5 * (row + 2) * - col)}]\sigma && \mathcal{V}_E(1) \\
&= \mathcal{V}_F[\texttt{5}]\sigma \times \mathcal{V}_T[\texttt{(row + 2) * - col)}]\sigma && \mathcal{V}_T(2) \\
&= 5 \times \mathcal{V}_T[\texttt{(row + 2) * - col)}]\sigma && \mathcal{V}_F(1) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times \mathcal{V}_T[\texttt{- col)}]\sigma\big) && \mathcal{V}_F(1) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times \mathcal{V}_F[\texttt{- col)}]\sigma\big) && \mathcal{V}_T(1) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_E[\texttt{col)}]\sigma\big) && \mathcal{V}_F(2b) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_T[\texttt{col)}]\sigma\big) && \mathcal{V}_E(1) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\mathcal{V}_F[\texttt{col)}]\sigma\big) && \mathcal{V}_T(1) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-\sigma(\texttt{col}))\big) && \mathcal{V}_F(1b) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{(row + 2)}]\sigma \times (-8)\big) && (\texttt{col}, 8) \in \sigma \\
&= 5 \times \big(\mathcal{V}_E[\texttt{row + 2}]\sigma \times (-8)\big) && \mathcal{V}_F(2a) \\
&= 5 \times \big(\mathcal{V}_T[\texttt{row}]\sigma + \mathcal{V}_E[\texttt{2}]\sigma \times (-8)\big) && \mathcal{V}_E(2a) \\
&= 5 \times \big(\mathcal{V}_F[\texttt{row}]\sigma + \mathcal{V}_T[\texttt{2}]\sigma \times (-8)\big) && \mathcal{V}_T(1),\ \mathcal{V}_E(1) \\
&= 5 \times \big(\sigma(\texttt{row}) + \mathcal{V}_F[\texttt{2}]\sigma) \times (-8)\big) && \mathcal{V}_F(1b),\ \mathcal{V}_T(1) \\
&= 5 \times ((5+2) \times (-8)) && (\texttt{row}, 5) \in \sigma,\ \mathcal{V}_F(1a) \\
&= -280 && \text{arithmetic}
\end{aligned}
$$

□

Exercises 8.4

1. Add a division operation symbol '`/` ' to the language E in Figure 8.4 in such a way that 'T/' and '`*` ' have equal precedence but there is no ambiguity.

2. Add a division operation symbol '`/` ' to the language E in Figure 8.4 in such a way that division operations are performed from right to left. Can right-to-left division and left-to-right multiplication have equal precedence?

Let $V = \mathbb{N} \cup \text{IDE} \cup \{\texttt{(}, \texttt{)}, \texttt{+}, \texttt{-}, \texttt{*}\}$. Simultaneously define (see Exercise 5, Section 5) the languages F, T, E and interpretations $\mathcal{V}_F, \mathcal{V}_T, \mathcal{V}_E$ as follows:

	$F \subseteq V^+$	$\mathcal{V}_F : F \times \text{ENV} \to \mathbb{N}$
1a.	$\mathbb{N} \subseteq F$	$\mathcal{V}_F[k]\sigma = k, \text{ for } k \in \mathbb{N}$
1b.	$\text{IDE} \subseteq F$	$\mathcal{V}_F[v]\sigma = \sigma(v), \text{ for } v \in \text{IDE}$
2a.	$e \in E \Rightarrow \texttt{(}\ e\ \texttt{)} \in F$	$\mathcal{V}_F[\texttt{(}\ e\ \texttt{)}]\sigma = \mathcal{V}_E[e]\sigma$
2b.	$e \in E \Rightarrow \texttt{-}\ e \in F$	$\mathcal{V}_F[\texttt{-}\ e]\sigma = -\mathcal{V}_E[e]\sigma$
3.	nothing else	

	$T \subseteq V^+$	$\mathcal{V}_T : T \times \text{ENV} \to \mathbb{N}$
1.	$F \subseteq T$	$\mathcal{V}_T[f]\sigma = \mathcal{V}_F[f]\sigma$ for $f \in F$
2.	$f \in F, t \in T \Rightarrow f\ \texttt{*}\ t \in T$	$\mathcal{V}_T[f\ \texttt{*}\ t]\sigma = \mathcal{V}_F[f]\sigma \times \mathcal{V}_T[t]\sigma$
3.	nothing else	

	$E \subseteq V^+$	$\mathcal{V}_E : E \times \text{ENV} \to \mathbb{N}$
1.	$T \subseteq E$	$\mathcal{V}_E[t]\sigma = \mathcal{V}_T[t]\sigma$ for $t \in T$
2a.	$t \in T, e \in E \Rightarrow t\ \texttt{+}\ e \in E$	$\mathcal{V}_E[t\ \texttt{+}\ e]\sigma = \mathcal{V}_T[t]\sigma + \mathcal{V}_E[e]\sigma$
2b.	$t \in T, e \in E \Rightarrow t\ \texttt{-}\ e \in E$	$\mathcal{V}_E[t\ \texttt{-}\ e]\sigma = \mathcal{V}_T[t]\sigma - \mathcal{V}_E[e]\sigma$
3.	nothing else	

Figure 8.1: A language of infix arithmetic expressions and its interpretation

3. Let environment $\sigma = \{(\mathtt{a}, 3), (\mathtt{b}, -2), (\mathtt{c}, 5)\}$. Evaluate one of the following words from the language E of Figure ?.

(a) `a + 6 - 3 * 5` (c) `b * 4 + - - y`
(b) `((b * 2) * c) * 3` (d) `a * - (b + 5)`

8.5 Backus-Naur Form

Languages defined in the manner of the previous sections are called *context free languages*. There is a standard notation in computer science for context free grammars. It is called *Backus-Naur form* or *BNF*, after John Backus and Peter Naur, who used it to specify the syntax of the algol 60 programming language. A BNF description of the language E in Figure 8.1 looks like this:

$$
\begin{array}{lll}
\langle \mathrm{F} \rangle & ::= & \langle \textsc{natural number} \rangle \\
\langle \mathrm{F} \rangle & ::= & \langle \textsc{program variable} \rangle \\
\langle \mathrm{F} \rangle & ::= & \mathtt{-}\ \langle \mathrm{E} \rangle \\
\langle \mathrm{F} \rangle & ::= & \mathtt{(}\ \langle \mathrm{E} \rangle\ \mathtt{)} \\
\langle \mathrm{T} \rangle & ::= & \langle \mathrm{F} \rangle \\
\langle \mathrm{T} \rangle & ::= & \langle \mathrm{F} \rangle\ \mathtt{*}\ \langle \mathrm{T} \rangle \\
\langle \mathrm{E} \rangle & ::= & \langle \mathrm{T} \rangle \\
\langle \mathrm{E} \rangle & ::= & \langle \mathrm{T} \rangle\ \mathtt{+}\ \langle \mathrm{E} \rangle \\
\langle \mathrm{E} \rangle & ::= & \langle \mathrm{T} \rangle\ \mathtt{-}\ \langle \mathrm{E} \rangle
\end{array}
$$

The names of inductively defined sets are surrounded by angle brackets $\langle \cdots \rangle$ and "$\langle L \rangle$ ::= *rule*" replaces "*rule* $\in L$. BNF allows us to describe languages concisely, without introducing variables for sub-phrases (e.g. f, t, and e in Figure 8.1).

Exercises 8.5

1. Give the BNF forms for the languages defined in Exercises 1 and 2.

2. Give the BNF form for the language of statements from Section 1.4. Assume that sets $\langle$*assignmentexpression*$\rangle$ and $\langle$*testexpression*$\rangle$ are already defined.

3. If you were going to define an interpretation function for the language of statements, what would its domain and range be?

8.6 Propositional Formulas

In this section we shall explore applications of the language definition style just introduced. For concreteness, the language of propositional logic is used. However, bear in mind that, except for the syntax, the results at the end of this section are valid for other languages, such as arithmetic expressions.

Figure 8.2 defines a language, PROP, of *propositional formulas*. It follows the style of Figure 8.1 in an abbreviated form:

(a) The language PROP is defined using BNF.

(b) Rather than building the language in stages, as was done with arithmetic terms, 8.2 simply specifies the operator precedence.

(c) The interpretation function $\mathcal{P}$ implicitly assumes that the language has been disambiguated.

It is supposed that the Reader can fill in the necessary details when needed.

In Figure 8.2, PROPs' meanings are given in terms of environments.

$$\text{ENV} = \{\sigma \mid \sigma\colon IVS \to \{T, F\}\}$$

A given environment $\sigma \in \text{ENV}$ corresponds to one row of a truth table. Accordingly, let us redefine *tautology*, *contradiction*, and *logical equivalence* in terms of environments.

Definition 8.1 *A* PROP *Q is a* tautology *iff for every $\sigma \in \text{ENV}$, $\mathcal{P}\sigma\,[Q] = T$. A* PROP *$Q$ is a* contradition *iff for every $\sigma \in \text{ENV}$, $\mathcal{P}\sigma\,[Q] = F$.*

Definition 8.2 *Two* PROP*s P and Q are* logically equivalent*, written $P \,\text{eq}\, Q$, iff for every $\sigma \in \text{ENV}$, $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$.*

As a first exercise, let us validate the intuitive correspondence between logical equivalence (eq) and bi-implication ($\Leftrightarrow$). Proposition 8.1 confirms a fact that we have already used. Conversely, its proof helps validate that our new definitions are sensible. For the purpose of this proposition, suppose an implication operator, `=>`, is included in PROP.

Proposition 8.1 *If P and Q are* PROP*s, then for all $\sigma \in \text{ENV}$*

$$P \text{ eq } Q \quad \text{iff} \quad \texttt{(}P \texttt{ => } Q\texttt{) \& (}Q \texttt{ => } P\texttt{)} \textit{ is a tautology.}$$

PROOF: The proof is a straightforward application of the definition of $\mathcal{P}$; induction is not required, although we do need to know that $\mathcal{P}$ is a function (not a relation or partial function).

Let PVAR be a set of propositional variables, and the alphabet $A = \text{PVAR} \cup \{\texttt{(},\ \texttt{)},\ \texttt{0}\ \texttt{1},\ \texttt{-},\ \texttt{|},\ \texttt{\&}\ >\}$. The language PROP $\subseteq A^+$ of *propositional formulas* and its interpretation

$$\begin{aligned} \text{ENV}: &\quad \text{PVAR} \longrightarrow \{\mathit{true}, \mathit{false}\} \\ \mathcal{P}: &\quad \text{ENV} \times \text{PROP} \longrightarrow \{\mathit{true}, \mathit{false}\} \end{aligned}$$

are defined as follows:

$\langle\text{PROP}\rangle$	$::=$	$\texttt{0}$	$\mathcal{P}_\sigma[\,\texttt{0}\,] = \mathit{false}$
	$\mid$	$\texttt{1}$	$\mathcal{P}_\sigma[\,\texttt{1}\,] = \mathit{true}$
	$\mid$	$\langle\text{PVAR}\rangle$	$\mathcal{P}_\sigma[\,v\,] = \sigma(v)$, for $v \in$PVAR
	$\mid$	$\texttt{-}\,\langle\text{PROP}\rangle$	$\mathcal{P}_\sigma[\,\texttt{-}\,f\,] = \neg\,\mathcal{P}_\sigma[\,f\,]$
	$\mid$	$\texttt{(}\,\langle\text{PROP}\rangle\,\texttt{)}$	$\mathcal{P}_\sigma[\,\texttt{(}\,f\,\texttt{)}\,] = \mathcal{P}_\sigma[\,f\,]$
	$\mid$	$\langle\text{PROP}\rangle\ \texttt{\&}\ \langle\text{PROP}\rangle$	$\mathcal{P}_\sigma[\,f_1\,\texttt{\&}\,f_2\,] = \mathcal{P}_\sigma[\,f_1\,] \wedge \mathcal{P}_\sigma[\,f_2\,]$
	$\mid$	$\langle\text{PROP}\rangle\ \texttt{|}\ \langle\text{PROP}\rangle$	$\mathcal{P}_\sigma[\,f_1\,\texttt{|}\,f_2\,] = \mathcal{P}_\sigma[\,f_1\,] \vee \mathcal{P}_\sigma[\,f_2\,]$

with precedence $\boxed{-} \succ \boxed{\&} \succ \boxed{|}$.

Figure 8.2: A language of propositional formulas and its interpretation

IF PART: Let $\sigma \in$ ENV and assume that P eq Q, that is, by Definition 8.2, $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$. Then

$$\begin{array}{lll}
\multicolumn{2}{l}{\texttt{(}P\ T\texttt{=>}\ Q\texttt{) \& (}Q\ \texttt{=>}\ P\texttt{)}} & \\
= & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[Q]) \wedge (\mathcal{P}\sigma\,[Q] \Rightarrow \mathcal{P}\sigma\,[P]) & \text{(Defn. of } \mathcal{P}\text{)} \\
= & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[P]) \wedge (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[P]) & \text{(Assumption that } \mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]\text{)} \\
= & T & \text{(meanings of } \Rightarrow, \wedge\text{)}
\end{array}$$

Thus, by Defintion 8.1, `(`P `=>` Q`) & (`Q `=>` P`)` is a tautology.

ONLY-IF PART: Assume that `(`P `=>` Q`) & (`Q `=>` P`)` is a tautology, and let $\sigma \in$ ENV be any environment.

$$\begin{array}{lll}
T & = & \mathcal{P}\sigma\,[\texttt{(}P \texttt{=>} Q\texttt{) \& (}Q \texttt{=>} P\texttt{)}] \\
 & = & (\mathcal{P}\sigma\,[P] \Rightarrow \mathcal{P}\sigma\,[Q]) \wedge (\mathcal{P}\sigma\,[Q] \Rightarrow \mathcal{P}\sigma\,[P]) \qquad \text{(defn. } \mathcal{P}\text{)}
\end{array}$$

This equality can hold only if $\mathcal{P}\sigma\,[P] = \mathcal{P}\sigma\,[Q]$. Therefore, by Definition 8.2, P *eq* Q.

□

Example 8.6 A function that translates sentences from one language to another (possibly the same) language called a *transliteration*. We are often interested in whether and how transliterations change the meaning of the original sentence. This example applies this idea to define a generalization of DeMorgan's identity for boolean algebras.

The *DeMorgan Dual* of $f \in$ PROP is obtained by switching all 0's and 1's, +'s and *'s, and inserting a `-` just before every variable symbol.

For instance, the DeMorgan dual of `( a &  b) | (( c | 0) & ( -b |  a))`
is `(-a | -b) & ((-c & 1) | (--b & -a))`

Inspecting the DNFs of these formulas: $\left\{ \begin{array}{l} \bar{a}\,\bar{b}\,c + a\,\bar{b}\,c + a\,b\,\bar{c} + a\,b\,c \\ \bar{a}\,\bar{b}\,\bar{c} + \bar{a}\,b\,\bar{c} + \bar{a}\,b\,c + a\,\bar{b}\,\bar{c}. \end{array} \right\}$ reveals that they are negations of each other—they have no clauses in common and together contain all eight possible clauses.

(a) *Define a recursive function* $\mathbb{D}$: PROP $\rightarrow$ PROP *that gives the DeMorgan dual of any* $F \in$ PROP*.*

$$\begin{array}{rcl}
\mathbb{D}[\,\texttt{0}\,] & = & \texttt{1} \\
\mathbb{D}[\,\texttt{1}\,] & = & \texttt{0} \\
\mathbb{D}[\,v\,] & = & \texttt{-}\hat{}\,v \text{ for } v \in \mathit{IVS} \\
\mathbb{D}[\,\texttt{-}\ P\,] & = & \texttt{-}\hat{}\,\mathbb{D}[\,P\,] \\
\mathbb{D}[\,P\ \texttt{|}\ Q\,] & = & \mathbb{D}[\,P\,]\hat{}\,\texttt{\&}\,\hat{}\,\mathbb{D}[\,Q\,] \\
\mathbb{D}[\,P\ \texttt{\&}\ Q\,] & = & \mathbb{D}[\,P\,]\hat{}\,\texttt{|}\,\hat{}\,\mathbb{D}[\,Q\,]
\end{array}$$

(b) *Prove that the DeMorgan dual of a propositional formula is its logical negation:* For all $\sigma \in \textsc{env}$ and $F \in \textsc{prop}$, $\mathcal{P}_\sigma[\,\mathbb{D}[\,F\,]\,] = \neg\,\mathcal{P}_\sigma[\,F\,]$.

The proof is a straightforward induction on words in PROP. The crux of the argument is in the base case for variables, where for any $v \in \mathit{IVS}$ we have

$$\mathcal{P}_\sigma[\,\mathbb{D}[\,v\,]\,] = \mathcal{P}_\sigma[\,\texttt{-}\hat{\ }v\,] = \neg\mathcal{P}_\sigma[\,v\,]$$

All steps above are justfied by the definitions of $\mathcal{P}$ or $\mathbb{D}$. An example of the inductive cases, for formulas of the form $P \texttt{ + } Q$, is

$$\begin{array}{rcll}
\mathcal{P}_\sigma\big[\mathbb{D}[\,P \texttt{ | } Q\,]\big] & = & \mathcal{P}_\sigma\big[\mathbb{D}[\,P\,]\hat{\ }\texttt{\&}\hat{\ }\mathbb{D}[\,Q\,]\big]\big] & \text{(defn. } \mathbb{D}) \\
& = & \mathcal{P}_\sigma\big[\mathbb{D}[\,P\,]\big] \wedge \mathcal{P}_\sigma\big[\mathbb{D}[\,Q\,]\big]\big] & \text{(defn. } \mathcal{P}) \\
& \overset{H}{=} & \neg\mathcal{P}_\sigma[\,P\,] \ \wedge\ \neg\mathcal{P}_\sigma[\,Q\,] & \text{(I.H. used twice)} \\
& = & \neg\,(\mathcal{P}_\sigma[\,P\,]\ \vee \mathcal{P}_\sigma[\,Q\,]) & \text{(DeMorgan's Identity)} \\
& = & \neg\mathcal{P}_\sigma[\,P\hat{\ }\texttt{|}\hat{\ }Q\,] & \text{(defn. } \mathcal{P})
\end{array}$$

□

□

Exercises 8.6

1. Add an implication operator, '`=>`' to PROP.

2. Use transliteration to add an implication "macro" to PROP. That is, define a language $\textsc{prop}_\Rightarrow$ that includes '`=>`' and a translation function $F\colon \textsc{prop}_\Rightarrow \to \textsc{prop}$ that correctly re-interprets f_1 `=>` f_2 as `(-` f_1 `+` f_2 `)` .

8.7 Substitution

A *substitution* is the simultanesous replacement of formulas for variables in an expression.

Definition 8.3 *Let* $F, P_1, \ldots, P_k \in \textsc{prop}$ *and* $v_1, \ldots, v_k \in \textsc{Pvar}$*. The* substitution

$$F\Big[{}^{P_1,\ \ldots,\ P_k}_{v_1,\ \ldots,\ v_k}\Big]$$

denotes the result, $\mathcal{S}[\,F\,]$*, of a substitution function* $S\colon \textsc{prop} \to \textsc{prop}$ *defined* *as follows:*

$$\begin{array}{rcl}
\mathcal{S}[\,\texttt{0}\,] & = & 0 \\
\mathcal{S}[\,\texttt{1}\,] & = & 1 \\
\mathcal{S}[\,x\,] & = & \begin{cases} P_j & \textit{if } x = v_j \\ x & \textit{otherwise} \end{cases} \\
\mathcal{S}[\,\texttt{-}\ Q\,] & = & \texttt{-}\hat{\ }\mathcal{S}[\,P\,] \\
\mathcal{S}[\,Q\ \texttt{|}\ Q'\,] & = & \mathcal{S}[\,Q\,]\hat{\ }\texttt{|}\hat{\ }\mathcal{S}[\,Q'\,] \\
\mathcal{S}[\,Q\ \texttt{\&}\ Q'\,] & = & \mathcal{S}[\,Q\,]\hat{\ }\texttt{\&}\hat{\ }\mathcal{S}[\,Q'\,]
\end{array}$$

According to the definition, we often consider the substitution *function*

$$\mathcal{S} \text{ as specified by } \left[{}^{P_1,\ \dots,\ P_k}_{v_1,\ \dots,\ v_k}\right]$$

which may be applied to any formula $F \in$ PROP, writing write $\mathcal{S}[\,P\,]$ to denote the result.

Example 8.7

$$\texttt{p * (-q + r)}\left[{}^{\texttt{q (s+t) p}}_{\texttt{p}\quad\texttt{q}\quad\texttt{r}}\right] \equiv \texttt{q * ((s+t) + p)}$$

In performing substitutions, one must take care to preserve operator precedence. Above, this is done by parenthesizing `(s + t)`.

□

The three theorems that follow verify the fundamental rules for logical manipulations involving substitution. Since we have been using these results all our lives, another way to look at these theorems is that they validate the definitions given so far.

Lemma 8.2 (Substitution Lemma) *Let $\mathcal{S}$ be a substitution and σ an environment. Define an new environment σ' as follows:*

$$\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}(v)] \text{ for } v \in \mathit{IVS}$$

Then for all PROP*s P,*

$$\mathcal{P}\sigma'\,[P] = \mathcal{P}\sigma\,[\mathcal{S}\,[P]]$$

PROOF: The proof is a straightforward structural induction on PROP. The inductive cases hold because the operations are functions. The interesting base case is the one for a variable $v \in \mathit{IVS}$. In that case, we have

$$\mathcal{P}\sigma'\,[v] = \mathcal{P}\sigma\,[\mathcal{S}\,[p]]$$

which is exactly what we need to make the Theorem true. □

The Substitution Lemma states that for the language of propositions, a call-by-value style evaluation—in which variables are bound to expressions' values—is equivalent to a call-by-name style evaluation. This is an exact equivalence because there is no form of looping in the language.

More significantly, the lemma says that our notion of substitution interacts well with our notion of evaluation. For example,

Theorem 8.3 (Tautology Theorem) *Let P be a* PROP *and $\mathcal{S}$ a substitution, If P is a tautology, then so is $\mathcal{S}[P]$.*

PROOF: Apply the definition of *tautology* and the Substitution Lemma. The details are left as Exercise **??**. $\square$

For example, the formula

$$Q \equiv \texttt{(p | q \& (p => r)) | -(p | q \& (p => r))}$$

is a tautology because `p | -p` is a tautology and there is a subsitition,

$$\mathcal{S}[\texttt{p}] = \texttt{(p | q \& (p => r))}$$

under which

$$Q \equiv \mathcal{S}[\texttt{p | -p}]$$

Theorem 8.3 gives us one way to abbreviate the analysis of PROPs. The next theorem states that we can analyze PROP *schemes* as well as individual PROPs.

Theorem 8.4 (Substitution Theorem) *If $P \,\mathrm{eq}\, Q$ then for any substitution $\mathcal{S}$, $\mathcal{S}[P] \,\mathrm{eq}\, \mathcal{S}[Q]$*

PROOF: Use the Substitution Lemma. $\square$

Thus, not only is formula

$$\texttt{p | (q | r)} \ \mathrm{eq}\ \texttt{(p | q) | r}$$

but the two formula *schemes*,

$$P \texttt{ | (}Q\texttt{ | }R\texttt{)} \quad \text{and} \quad \texttt{(}P\texttt{ | }Q\texttt{) |}R$$

are equivalent for arbitray sub-PROPs P, Q, and R. In fact, we reason about PROP schemes far more often than we reason about PROP individuals.

The following result is very important to the way we do proofs. It says you can "replace equals with equals" and still preserve equivalence.

Theorem 8.5 (Replacement Theorem) *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be substitutions such that, for all $v \in IVS$, $\mathcal{S}_1(v)$ eq $\mathcal{S}_2(v)$. Then for any* PROP *P,*

$$\mathcal{S}_1[P] \ \mathrm{eq}\ \mathcal{S}_2[P]$$

PROOF: Let σ be any environment and define σ' to be

$$\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}_1[v]]$$

Since $\mathcal{S}_1(v)$ eq $\mathcal{S}_2(v)$, it is also the case that $\sigma'(v) = \mathcal{P}\sigma\,[\mathcal{S}_2[v]]$ for all $v \in \mathit{IVS}$. Using the Substitution Lemma twice, we have

$$\mathcal{P}\sigma\,[\mathcal{S}_1[P]] = \mathcal{P}\sigma'\,[P] = \mathcal{P}\sigma\,[\mathcal{S}_2[P]]$$

as desired. $\square$

For example, `p⇒q` eq `¬q ⇒ ¬p` so

$$\texttt{(q} \Rightarrow \texttt{p)} \wedge \texttt{(p} \Rightarrow \texttt{q)} \quad \text{eq} \quad \texttt{(q} \Rightarrow \texttt{p)} \wedge \texttt{(}\neg\texttt{q} \Rightarrow \neg\texttt{p)}$$

under the substitutions

v	$\mathcal{S}_1[v]$	$\mathcal{S}_2[v]$
`p`	`p`	`p`
`q`	`q`	`q`
`r`	`p ⇒ q`	`¬q ⇒ ¬p`

These results about substitution and replacement do not depend in any fundamental way on the syntax of formulas in PROP. Exercise **??** asks you to define substitution for arithmetic expressions. Exercise 3 asks you to consider a more general definition of substitution, applicable to any language defined in the style developed in this chapter.

The question of interpretation is more important. Is it the case that all recursively defined interpretations allow substitution? One answer is that substitution is so important that only those interpretations that make the Substitution Lemma (Lemma 8.2) true are allowed.

Exercises 8.7

1. Let $F \equiv p \wedge (q \vee r)$. Perform the following substitutions

(a) $F\left[\begin{smallmatrix} r, & q, & p \\ p, & q, & r \end{smallmatrix}\right]$

(b) $F\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]$

(c) $F\left[\begin{smallmatrix} p \vee r, & q \Rightarrow r \\ p, & r \end{smallmatrix}\right]$

(d) $\left(F\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]\right)\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right]$

(e) $\left(F\left[\begin{smallmatrix} q \\ p \end{smallmatrix}\right]\right)\left[\begin{smallmatrix} p \vee r \\ p \end{smallmatrix}\right]$

2. Define substitution for arithmetic terms as defined in Figure 8.1. Show that the Substitution Lemma (Lemma 8.2) holds for their interpretation.

3. Describe in general terms the process of defining substitution for any language that contains variables. Try to write down the appropriate generalized definitions and theorems.

8.8 The Programming Language of Statements

The STMT programming language was first introduced in Chapter 1 and has been referred to often throughout this textbook. In this section we shall apply the definitional style developed in this chapter to write a more rigorous specification of program syntax and meaning. These specifications appear in Definitions 8.4 and 8.5.

Sentences in STMT—programs—are built from a set of keywords,

$$\{\texttt{begin}, \texttt{end}, \texttt{if}, \texttt{then}, \texttt{else}, \texttt{while}, \texttt{do}, \texttt{:=}, \texttt{;}\}$$

and phrases coming from:

(a) The language of *arithmetic terms* used in assignment statements. This language and its interpretation are essentially the same as that of Figure 8.1.

(b) *Test expressions* used in `if` and `while` statements. Test expressions are logical combinations of arithmetic comparisons. Their interpretation is straightforward to define and is left as an exercise.

Discussion Points The clauses in Definition 8.5 are subtle. They must be studied carefully.

- Programs express computation in terms of *assignment*: recording information in a memory. The interpretation of statements reflects this model. A program starts with an initial memory, runs for a while, and then stops, leaving its results in an updated memory. We model memories abstractly with environments mapping program variables to values. Thus, the interpretation function maps from environments to environments.

  $$\mathcal{M} \colon \text{ENV} \times \text{STMT} \xrightarrow{p} \text{ENV}$$

  $\mathcal{M}$ is a *partial* function. For non-terminating programs it does not give a value, as discussed below.

- The interpretation of assignment says to take the environment function σ, remove the ordered pair for program variable V, and replace it with one that binds V to the value of T.

  $$[\sigma \setminus \{(V, \sigma(V)\}] \cup \{(v, \mathcal{T}_\sigma[\,T\,]\}$$

- The compound-statement interpretations says, execute statement S_1, and then execute S_2 using this resulting memory, σ'. It might have been written even more mysteriously as
$$\mathcal{M}_{(\mathcal{M}_\sigma[\,S_1\,])}[\,S_2\,]$$
- Unlike all other rules, the rule for `while`-statements does not reduce interpretation to a *proper* sub-sentence. The interpretation of certain statements is undefined. For instance,
$$\begin{array}{rl} & \mathcal{M}_\sigma[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ & \quad \boxed{\mathcal{P}_\sigma[\,x = x\,] = \cdots = \mathit{true}} \\ \overset{(d)}{==} & \mathcal{M}_{\sigma'}[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ & \quad \boxed{\text{where } \sigma'(x) = \mathcal{M}_\sigma[\,x := x\,] = \cdots = \sigma} \\ = & \mathcal{M}_\sigma[\,\texttt{while}\ x = x\ \texttt{do}\, x := x\,] \\ \vdots & \end{array}$$
Of course, this is just what we want our model to describe: a non-terminating program does not produce a "final" memory.

Let us use Definition 8.5 prove an interesting fact about compound statements.

Proposition 8.6 *The compound operator, '`;`' is associative in the sense that for all $\sigma \in$ ENV and statements S_1, S_2, and S_3,*
$$\begin{array}{rl} & \mathcal{M}_\sigma[\,\texttt{begin begin}\ S_1\ \texttt{;}\ S_2\ \texttt{end ;}\ S_3\ \texttt{end}\,] \\ = & \mathcal{M}_\sigma[\,\texttt{begin}\ S_1\ \texttt{; begin}\ S_2\ \texttt{;}\ S_3\ \texttt{end end}\,] \end{array}$$

PROOF: Apply Definition 8.5. □

Thus, it is not ambiguous to write `begin` S_1 `;` S_2 `;` S_3 `end` because it doesn't matter how `begin`-`ends` are associated.

The next relatively simple fact is the first step in reducing program correctness statements to purely logical conditions. Recall that the notation $\{P\}$ `S` $\{Q\}$ says, "If statement S starts with a memory in which property P holds (and if it terminates), property Q holds in the final memory. In more precise terms, for all $\sigma \in$ ENV,
$$\mathcal{P}_\sigma[\,P\,] \text{ implies } \mathcal{P}_{\sigma'}[\,Q\,] \text{ where } \sigma' = \mathcal{M}_\sigma[\,S\,]$$

Proposition 8.7 $\{P\}\,V$ `:=` $T\,\{Q\}$ *iff* $P \Rightarrow Q^T_V$.

PROOF: ($\Rightarrow$) Let $\sigma \in$ ENV, and assume $\{P\}\,V$ `:=` $T\,\{Q\}$ is true. Then $\mathcal{P}_\sigma[\,P\,] \Rightarrow \mathcal{P}_{\sigma'}[\,Q\,]$ where $\sigma' = \mathcal{M}_\sigma[\,V\,\texttt{:=}\,T\,]$. By Definition 8.5, $\sigma'(V) = \mathcal{T}_\sigma[\,T\,]$. Hence, by the Substitution Lemma (Lemma 8.2, suitably generalized), $\mathcal{P}_{\sigma'}[\,Q\,]$ is logically equivalent to $\mathcal{P}_\sigma[\,Q^T_V\,]$. Therefore, $\mathcal{P}_\sigma[\,P \Rightarrow Q^T_V\,]$ is true.

($\Rightarrow$) By the same argument as above. □

Definition 8.4 *The languages of arithmetic terms (*TERM*), comparisons (*COMP*), tests (*TEST*) and program statements (*STMT*) are defined according to the grammar below. All operators associate to the right with precedence* $\boxed{-}_1 \succ \boxed{*} \succ \boxed{+} = \boxed{-}_2$ *and*

$$
\begin{array}{lll}
\langle\text{TERM}\rangle & ::= & \langle\text{NUMERAL}\rangle \\
 & ::= & \langle\text{IDENTIFIER}\rangle \\
 & ::= & \texttt{-}_1\ \langle\text{TERM}\rangle \\
 & ::= & \texttt{(}\ \langle\text{TERM}\rangle\ \texttt{)} \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{*}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{+}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle\ \texttt{-}_2\ \langle\text{TERM}\rangle
\end{array}
\qquad
\begin{array}{lll}
\langle\text{COMP}\rangle & ::= & \langle\text{TERM}\rangle\ \texttt{=}\ \langle\text{TERM}\rangle \\
 & ::= & \langle\text{TERM}\rangle < \langle\text{TERM}\rangle \\
\langle\text{TEST}\rangle & ::= & \langle\text{COMP}\rangle \\
 & ::= & \texttt{-}_3\ \langle\text{COMP}\rangle \\
 & ::= & \texttt{(}\ \langle\text{COMP}\rangle\ \texttt{)} \\
 & ::= & \langle\text{COMP}\rangle\ \&\ \langle\text{COMP}\rangle \\
 & ::= & \langle\text{COMP}\rangle\ \texttt{|}\ \langle\text{COMP}\rangle
\end{array}
$$

$$
\begin{array}{llll}
\langle\text{STMT}\rangle & ::= & \langle\text{IDENTIFIER}\rangle\ \texttt{:=}\ \langle\text{TERM}\rangle & \textit{(}\text{assignment}\textit{)} \\
 & ::= & \texttt{begin}\ \langle\text{STMT}\rangle\ \texttt{;}\ \langle\text{STMT}\rangle\ \texttt{end} & \textit{(}\text{compound}\textit{)} \\
 & ::= & \texttt{if}\ \langle\text{TEST}\rangle\ \texttt{then}\ \langle\text{STMT}\rangle\ \texttt{else}\ \langle\text{STMT}\rangle & \textit{(}\text{conditional}\textit{)} \\
 & ::= & \texttt{while}\ \langle\text{TEST}\rangle\ \texttt{do}\ \langle\text{STMT}\rangle & \textit{(}\text{repetition}\textit{)}
\end{array}
$$

Definition 8.5 *Given interpretations* $\mathcal{T}\colon \text{ENV} \times \text{TERM} \to \mathbb{N}$ *and* $\mathcal{P}\colon \text{ENV} \times \text{TEST} \to \{\textit{true}, \textit{false}\}$*, the operational meaning of programs in* STMT *(Definition 8.4) is given by the partial function* $\mathcal{M}\colon \text{ENV} \times \text{STMT} \to \text{ENV}$ *defined as follows:*

$$(a)\quad \mathcal{M}_\sigma[\,V\ \texttt{:=}\ T\,] = [\sigma \setminus \{(V, \sigma(V)\}] \cup \{(V, \mathcal{T}_\sigma[\,T\,]\}$$

$$(b)\quad \mathcal{M}_\sigma[\,\texttt{begin}\ S_1\ \texttt{;}\ S_2\ \texttt{end}\,] = \mathcal{M}_{\sigma'}[\,S_2\,] \text{ where } \sigma' = \mathcal{M}_\sigma[\,S_1\,]$$

$$(c)\quad \mathcal{M}_\sigma[\,\texttt{if}\ Q\ \texttt{then}\ S_1\ \texttt{else}\ S_2\,] = \begin{cases} \mathcal{M}_\sigma[\,S_1\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{true} \\ \mathcal{M}_\sigma[\,S_2\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{false} \end{cases}$$

$$(d)\quad \mathcal{M}_\sigma[\,\texttt{while}\ Q\ \texttt{do}\,S\,] = \begin{cases} \sigma, & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{false} \\ \mathcal{M}_{\sigma'}[\,\texttt{while}\ Q\ \texttt{do}\ S\,] & \text{if } \mathcal{P}_\sigma[\,Q\,] = \textit{true} \\ \quad \text{where } \sigma' = \mathcal{M}_\sigma[\,S\,], & \end{cases}$$

Example 8.8 In Proposition **??** (p. **??**) it was shown that

```
{$z + xy = AB$}
while $x \neq 0$ do
    begin
    $x := x - 1$;
    $z := z + y$
    end;
end {$z = AB$}
```

The proof of invariance involved reasoning about the values of identifiers before (x, y and z) and after (x', y' and z') executing the loop body. Proposition 8.7 says that this kind of temporal reasoning may be reduced to "pure logic":

$$\{z + xy = AB \ \wedge \ x \neq 0\}\,\texttt{begin}\ \ x \mathrel{\texttt{:=}} x - 1\,\texttt{;}\ \ z \mathrel{\texttt{:=}} z + 1\ \texttt{end}\,\{z + xy = AB\}$$

eq (by Prop. 8.7)

$$\{z + xy = AB \ \wedge \ x \neq 0\}\ x \mathrel{\texttt{:=}} x - 1\ \left\{(z + xy = AB)\left[{}^{z\,+\,1}_{\ z}\right]\right\}$$

eq (by Prop. 8.7)

$$\left(z + xy = AB \ \wedge \ x \neq 0\right) \ \Rightarrow \ \left((z + xy = AB)\left[{}^{z\,+\,y}_{\ z}\right]\right)\left[{}^{x\,-\,1}_{\ x}\right]$$

Performing these substitutions, we get

$$\left(z + xy = AB \ \wedge \ x \neq 0\right) \ \Rightarrow \ \left((z + xy = AB)\left[{}^{z\,+\,y}_{\ z}\right]\right)\left[{}^{x\,-\,1}_{\ x}\right]$$

eq

$$\left(z + xy = AB \ \wedge \ x \neq 0\right) \ \Rightarrow \ \left((z + y) + xy = AB\right)\left[{}^{x\,-\,1}_{\ x}\right]$$

eq

$$\left(z + xy = AB \ \wedge \ x \neq 0\right) \ \Rightarrow \ \left[(z + y) + (x - 1)y = AB\right]$$

eq (simplifying $(x + y) + (x - 1)y$)

$$\left(z + xy = AB \ \wedge \ x \neq 0\right) \ \Rightarrow \ \left(z + xy = AB\right)$$

Which is tautologically true.

□

Exercises 8.8

1. Define interpretations for arithmetic and test expressions. $\mathcal{T}$: ENV × TERM → $\mathbb{N}$ $\mathcal{P}$: ENV × TEST → {*true*, *false*}.

2. In Example 8.8 the subformula

$$\Big(\big(z + xy = AB\big)\Big[{}^{z\,+\,y}_{z}\Big]\Big)\Big[{}^{x\,-\,1}_{x}\Big]$$

is derived. Is this the same as

$$\big(z + xy = AB\big)\Big[{}^{z\,+\,y,\;x\,-\,1}_{z,\qquad x}\Big]$$

3. Suppose that, for all $\sigma \in$ ENV, $T \in$ TERM and $C \in$ COMP, $\mathcal{T}_\sigma[\,T\,] = 42$ and $\mathcal{P}_\sigma[\,C\,] = \mathit{true}$. Describe how programs in STMT behave.

8.9 *Discussions

8.9.1 Parenthesized Expressions

In Example 8.2 it is claimed that fully parenthesized expressions are unambiguous. This fact seems intuitively obvious but proving it rigourously requires delving into that intuition. The essential property of this language is that its parentheses are "balanced."

As in the example, Let alphabet $A = \mathbb{N} \cup \{\texttt{\#}, \texttt{\$}, \texttt{(}, \texttt{)}\}$. For $w \in A^+$, define $\Delta[w]$ to be the number of left parentheses in w minus the number of right parentheses. For example, $\Delta[\texttt{((}5\,\texttt{\$}\,2\,\texttt{()}] = 2$ and $\Delta[\texttt{)}3\texttt{()}7\texttt{(}] = 0$.

The language L_2 of 8.2 was defined

	$L_2 \subseteq A^+$			$\mathcal{V}\colon L_2 \to \mathbb{N}$
1.	$\langle L_2 \rangle$	$::=$	$\mathbb{N}$	$\mathcal{V}[\,n\,] = n$ for $n \in \mathbb{N}$
2*a*.		$\mid$	$\texttt{(}\langle L_2\rangle\,\texttt{\#}\,\langle L_2\rangle\texttt{)}$	$\mathcal{V}[\,\texttt{(}\,u\,\texttt{\#}\,v\,\texttt{)}\,] = \mathcal{V}[\,u\,] + \mathcal{V}[\,v\,]$
2*b*.		$\mid$	$\texttt{(}\langle L_2\rangle\,\texttt{\$}\,\langle L_2\rangle\texttt{)}$	$\mathcal{V}[\,\texttt{(}\,u\,\texttt{\$}\,v\,\texttt{)}\,] = \mathcal{V}[\,u\,] \times \mathcal{V}[\,v\,]$

Proposition 8.8 *Let L_2 be the language defined in Example 8.2. Then for all $w \in L_2$, $\Delta[w] = 0$.*

Proof: The proof is by structural induction on L_2. In the base case, if $k \in \mathbb{N}$ then k contains no parentheses, so $\Delta[k] = 0$. For the induction case, assume $\Delta[u] = \Delta[v] = 0$. Then

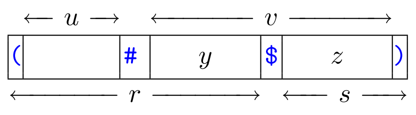
$$\Delta[\,\texttt{(}\,u\,\texttt{*}\,v\,\texttt{)}\,] = 1 + \Delta[u] + \Delta[v] - 1 = 0$$

And similarly for $\Delta[\,\texttt{(}\,u\,\texttt{+}\,v\,\texttt{)}\,]$. □

Proposition 8.8 captures only part of the quality of being balanced. How can we capture the notion of being *properly* balanced? The answer is not obvious, but a little experimentation will convince you that the next proposition is what we need:

Proposition 8.9 *Let L_2 be the language defined in Example 8.2, and let $w \in L_2$. If $w = r\,\texttt{\#}\,s$ (or $r\,\texttt{\$}\,s$) for any two words $r, s \in V^+$, then $\Delta[r] > 0$ and $\Delta[s] < 0$.*

Proof: The proof is by structural induction on L_2 and the base case holds vacuously. For the induction case, assume $w = \texttt{(}\,u\,\texttt{\#}\,v\,\texttt{)}$ and that the induction hypothesis holds for u and v. Now suppose that w also equals $r\,\texttt{\$}\,s$, so that we have the following:



By Proposition 8.8 $\Delta[u] = 0$, and by the induction hypothesis, $\Delta[y] > 0$. Hence,

$$\Delta[r] = \Delta[\texttt{(}u \texttt{ \# } y] = 1 + \Delta[y] > 0$$

Again by Proposition 8.8, $\Delta[w] = 0$, so it must be the case that

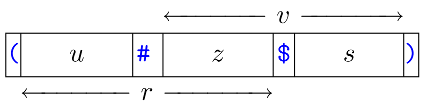
$$\Delta[s] = \Delta[z\texttt{)}] = -\Delta[r] < 0$$

A similar argument holds if the '$' occurs to the right of the '#'. □

We are now in a position to prove that each word in L_2 has a unique parse.

Proposition 8.10 *Let L_2 be the language defined in Example 8.2, and let $w \in L_2$, $w \notin \mathbb{N}$. Then there is exactly one pair of words, $u, v \in L_2$ such that $w =$ (u # v) (or (u $ v)).*

Proof: We will assume that there are two such pairs and reach a contradiction. Without loss in generality, assume that $w =$ (u # v) and $w =$ (r $ s) and u, v, r, $s \in L_2$, so that we have the following:



By Proposition 8.8, $r \in L_2$ implies $\Delta[z] < 0$ while $u \in L_2$ implies $\Delta[z] > 0$. This is a contradiction, so either u and v or r and s do not exist. □

Index