Chapter 8

Inductive and Recursive Definitions

Great fleas have little fleas,
Upon their backs to bite 'em,
And little fleas have lesser fleas,
And so ad infinitum.

- Augustus De Morgan.

Most of the objects under study within Computer Science are defined *inductively*: that is, they are defined in terms of smaller instances of themselves. Numbers, lists, binary trees, and even computer programs themselves, are all built up from smaller objects of the same type. For example, two computer programs stuck together, typically with a semicolon between them, so that the second is executed once the first completes its task is nothing more than a program defined in terms of two smaller programs. Also, functions defined over such objects are typically given by inductive definitions, whereby the value of the function on an inductively-defined object is defined by the value of the function on smaller objects. More generally, a *recursive* definition allows a function to be defined in terms of its value on arbitrary objects, not necessarily smaller objects, and can be meaningfully employed.

Understanding inductively-defined objects, and the functions defined on them, will naturally rely on understanding the inductive nature of such definitions. In this chapter, we explore such inductive definitions and recursively-defined functions.

8.1 Inductively-Defined Sets

As we saw, we can define finite sets by simply listing their elements, such as

$$\begin{aligned}
\textsc{BinaryDigits} &= \{0, 1\}\\
\textsc{DecimalDigits} &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\\
\textsc{Letters} &= \{\,\mathrm{a, b, c, d, e, f, g, h, i, j, k, l, m,}\\
&\qquad \mathrm{n, o, p, q, r, s, t, u, v, w, x, y, z}\,\}\\
\mathsf{Children} &= \{\text{Joel, Felix, Oskar, Amanda}\}
\end{aligned}$$

However, for infinite sets we have had to resort to using some (implicit or explicit) rule for generating their members. For example, the set of natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$$

which we defined (informally) in Chapter 2 relies on our ability as intelligent beings to extract the implicit rule hinted at by the ellipses which says that adding one to any element of this set gives the "next" element in the set. However, this approach to defining sets is fraught with complications.

1. How can we expect a non-intelligent entity (such as a computer) to be able to understand such a definition? At the very least we would somehow have to make explicit the rule for generating the elements of the set.
2. How can we even be certain of the implicit rule underlying the defining equation? For example, the author of the above definition may intend $\mathbb{N}$ to represent the decimal digits (and thus end at the digit 9), or the roots (i.e., solutions) of the equation $x^4 - 6x^3 + 11x^2 - 6x = 0$ (in which case $\mathbb{N}$ would contain only the four values listed).
3. The order in which we list the elements of a set is irrelevant, so what sense does it make to refer to the "next" element in a set?
4. How can we determine when some object, $\sqrt{9}$ say, is in the set we are defining while another object, $\sqrt{10}$ say, is not?

One easy way of defining an infinite collection of objects is to provide a method for generating new elements from existing ones. This idea is encompassed by the following definition.

Definition 8.1

*An **inductive definition** of a set has three components.*

1. *The **basis clause**, which establishes that certain objects are in the set. These elements constitute the "building blocks" for constructing further elements in the set.*
2. *The **inductive clause**, which defines the ways in which elements of a set can be used to produce further elements which are also in the set.*

3. *The **extremal clause**, which asserts that no object is an element of the set being defined unless its membership can be established from the first two clauses. In other words, the set being defined is the **smallest** set which satisfies the first two clauses.*

Example 8.1

We can represent precisely the set $\mathbb{N}$ of natural numbers by way of the following inductive definition.

1. $0 \in \mathbb{N}$.
2. $(n+1) \in \mathbb{N}$ whenever $n \in \mathbb{N}$.

   In other words, $n \in \mathbb{N} \;\Rightarrow\; (n+1) \in \mathbb{N}$.
3. Nothing else is in $\mathbb{N}$. That is, nothing is in $\mathbb{N}$ unless it can be constructed from the first two clauses.

   In other words, $\mathbb{N}$ is the *smallest* set satisfying the first two clauses.

The basis clause declares the number 0 as a basic element of the set $\mathbb{N}$; and the inductive clause says that given a natural number n, we can produce another natural number $n+1$ by adding 1 to the given number n. In this way we can conclude that $\sqrt{9} = 3$ is an element of $\mathbb{N}$, since 0 is an element of $\mathbb{N}$ (by the basis clause), and hence $0+1 = 1$ is an element (by the inductive clause), and hence $1+1 = 2$ is an element (again by the inductive clause), and thus finally $2+1 = 3$ is an element (by a further use of the inductive clause).

The extremal clause tells us that an element of $\mathbb{N}$ has to be either 0 (from the basis clause) or the successor of another element of $\mathbb{N}$ (from the inductive clause). We could not infer that $\sqrt{10} \approx 3.16$ is an element of $\mathbb{N}$, as there is no way to construct $\sqrt{10}$ from these basis and inductive clauses: $\sqrt{10}$ is clearly not 0; and no matter how many times we add 1 to 0 we will never generate the value $\sqrt{10}$. Hence we must conclude that $\sqrt{10}$ is *not* an element of $\mathbb{N}$ as defined.

Alternatively, we can easily see that the set $\{0, 1, 2, 3, 4, \ldots\}$ satisfies clauses (1) and (2) of the definition. Therefore, since $\mathbb{N}$ is being defined to be the *smallest* set satisfying these clauses, $\mathbb{N}$ must be a subset of this; since this set does not contain $\sqrt{10}$, $\sqrt{10} \notin \mathbb{N}$.

Exercise 8.1 (Solution on page 440)

Explain, using this inductive definition of $\mathbb{N}$, why $4 \in \mathbb{N}$ while $4.5 \notin \mathbb{N}$.

Example 8.2

We can inductively define the set

$$\textsc{Odd} = \{1, 3, 5, 7, \ldots\}$$

of odd natural numbers as the smallest set satisfying the following:

1. $1 \in \textsc{Odd}$.
2. If $n \in \textsc{Odd}$ then $(n+2) \in \textsc{Odd}$.

Note that in this example, we incorporated the extremal clause into the preamble of the definition, by defining the set to be the smallest set satisfying the basis and inductive clauses; being the smallest such set, only those elements which must be in the set due to the basis and inductive clauses are actually members. We could have instead included the extremal clause; however, the above is a common useful abbreviated form.

Exercise 8.2 (Solution on page 441)

The set $\mathbb{N}$ satisfies the two clauses in the definition of $\textsc{Odd}$; that is, it contains 1, and it contains $(n+2)$ whenever it contains n. Why does this not imply that $\textsc{Odd} = \mathbb{N}$?

Exercise 8.3 (Solution on page 441)

Give an inductive definition for the set $\textsc{Powers-of-2}$ of powers of 2,

$$\textsc{Powers-of-2} = \{1, 2, 4, 8, 16, 32, 64, \ldots\}.$$

Example 8.3

Given a finite set S, we can define the powerset $\mathcal{P}(S)$ of S inductively as the smallest set satisfying the following:

1. $\emptyset \in \mathcal{P}(S)$.
2. If $X \in \mathcal{P}(S)$ and $a \in S$ then $X \cup \{a\} \in \mathcal{P}(S)$.

For example, if $S = \{1, 2, 3\}$, then by the basis clause $\emptyset \in \mathcal{P}(S)$, and by one application of the inductive clause we get that the following sets are in $\mathcal{P}(S)$:

$$\emptyset \cup \{1\} = \{1\} \qquad \emptyset \cup \{2\} = \{2\} \qquad \emptyset \cup \{3\} = \{3\}$$

This application reveals that all of the singleton sets $\{1\}$, $\{2\}$ and $\{3\}$ are in $\mathcal{P}(S)$. A second application of the inductive clause tells us that the following sets are in $\mathcal{P}(S)$:

$$\begin{array}{lll}
\emptyset \cup \{1\} = \{1\} & \emptyset \cup \{2\} = \{2\} & \emptyset \cup \{3\} = \{3\} \\
\{1\} \cup \{1\} = \{1\} & \{1\} \cup \{2\} = \{1, 2\} & \{1\} \cup \{3\} = \{1, 3\} \\
\{2\} \cup \{1\} = \{1, 2\} & \{2\} \cup \{2\} = \{2\} & \{2\} \cup \{3\} = \{2, 3\} \\
\{3\} \cup \{1\} = \{1, 3\} & \{3\} \cup \{2\} = \{2, 3\} & \{3\} \cup \{3\} = \{3\}
\end{array}$$

This second application reveals that all of the two-element sets $\{1, 2\}$, $\{1, 3\}$ and $\{2, 3\}$ are also in $\mathcal{P}(S)$. A third application of the inductive clause would reveal that, apart from the above sets, the three-element set $S = \{1, 2, 3\}$ itself is in $\mathcal{P}(S)$. Further applications of the inductive clause would generate no new elements.

Exercise 8.4 (Solution on page 441)

Why can the above definition not be applied to infinite sets? (Hint: Why would this definition not provide $\text{ODD} \in \mathcal{P}(\mathbb{N})$, where ODD is as defined in Example 8.2?)

8.2 Inductively-Defined Syntactic Sets

The elements of the set $\mathbb{N}$ of natural numbers as defined above are *semantic* values, not *syntactic* objects. To understand the distinction clearly, if we define the set

$$\text{Children} = \{\text{Joel, Felix, Oskar, Amanda}\}$$

we have to make clear whether we mean the set of four names, or the collection of people which make up four specific children. Each name in the list is merely a syntactic object unless we assign some meaning or semantic content to it.

In the same way, we have that $\sqrt{9}$ is an element of $\mathbb{N}$, as $\sqrt{9} = 3$ and 3 is an element of $\mathbb{N}$. The set $\mathbb{N}$ represents the collection of values making up the natural numbers, not some arbitrary representation of them such as decimal numbers (sequences of decimal digits) or binary numbers (sequences of binary digits).

To define sets of such syntactic objects, we first introduce some terminology. An ***alphabet*** is a finite set of ***symbols*** or ***characters***. A finite sequence of characters from an alphabet A is called a ***string*** or ***word*** over A. The ***length*** of a word $w = a_1a_2a_3 \cdots a_n$, where $n \in \mathbb{N}$ and $a_i \in A$ for each $1 \leq i \leq n$, is given by the number n of (occurrences of) characters in w. We shall use the special symbol ε (which cannot be a character of the alphabet A) to denote the ***empty word***, that is, the only word of length 0. Note that $\varepsilon w = w\varepsilon = w$ for any word w.

Finally, we shall use A^* to denote the set of all words over A, and A^+ to denote the set of non-empty words over A. We can define these two sets inductively as follows.

Definition 8.4

The set A^* of words over alphabet A is the smallest set satisfying the following:

1. $\varepsilon \in \boldsymbol{A}^*$; *and*
2. *if* $w \in A^*$ *and* $a \in A$ *then* $aw \in A^*$.

The set A^+ of non-empty words over alphabet A is the smallest set satisfying the following:

1. $a \in A^+$ *for each* $a \in A$; *and*
2. *if* $w \in A^+$ *and* $a \in A$ *then* $aw \in A^+$.

Example 8.4

If $A = \{a, b\}$, then A^* is the set consisting of all sequences of a's and b's, including the empty sequence containing no characters:

$$A^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \dots\}.$$

This is since:

- by the first (basis) clause, $\varepsilon \in A^*$;
- by the second (inductive) clause, adding either an a or a b to the front of any word in A^* gives a word in A^*, and as we know $\varepsilon \in A^*$, this means that $\{a, b\} \subseteq A^*$;
- but then by the second (inductive) clause, since we now know that $\{\varepsilon, a, b\} \subseteq A^*$, we can infer that $\{a, b, aa, ab, ba, bb\} \subseteq A^*$;
- by a third application of the second (inductive) clause, we can now infer that

$$\begin{aligned}\{\ & a, b, aa, ab, ba, bb,\\ & aaa, aab, aba, abb, baa, bab, bba, bbb\} \subseteq A^*;\end{aligned}$$

and each new application of the second (inductive) clause adds more new strings to the set.

Similarly, A^+ is the set consisting of all non-empty sequences of a's and b's:

$$A^+ = \{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \dots\}.$$

We could have defined the sets A^* and A^+ in various other equivalent ways. For example, we could have used wa instead of (or as well as) aw in each of the second (inductive) clauses; or we could have provided just one inductive definition and defined the second set directly in terms of the first, by observing that $A^* = A^+ \cup \{\varepsilon\}$ and $A^+ = A^* \setminus \{\varepsilon\}$.

We can now define the sets of decimal and binary numbers as the sets of non-empty words over decimal, respectively binary, digits.

$$\begin{aligned} \textsc{DecimalNumbers} &= \textsc{DecimalDigits}^+ \\ \textsc{BinaryNumbers} &= \textsc{BinaryDigits}^+ \end{aligned}$$

Exercise 8.5 (Solution on page 441)

Give an inductive definition of $\textsc{PosDecimalNumbers}$, the set of positive decimal numbers. Such numbers should not have leading zeros; that is, $35 \in \textsc{PosDecimalNumbers}$ but $035 \notin \textsc{PosDecimalNumbers}$.

8.3 Backus-Naur Form

A common style of presenting an inductive definition of a set of syntactic objects is the so-called ***Backus-Naur Form (BNF)***, in which the syntactic forms are presented equationally. For example, the set A^* of words over A is given by the BNF equation

$$w \ ::= \ \varepsilon \mid aw$$

and the set A^+ of non-empty words over A is given by the BNF equation

$$w \ ::= \ a \mid aw$$

where in both cases a is taken to range over the alphabet A. In this way, BNF provides a short-hand form of writing out inductive definitions.

As another example, the natural numbers $\mathbb{N}$ were defined in terms of zero 0 and the ***successor function*** $s(n) = n+1$. These elements can be specified by the BNF equation

$$n \ ::= \ 0 \mid s(n).$$

Hence, for example, the number 4 is formally defined as $s(s(s(s(0))))$.

Inductive definitions of sets of syntactic expressions are very common in Computer Science. Indeed we have seen several already, such as the set of propositional formulæ, which we can now define formally as follows.

Example 8.5

The set of propositional formulæ can be defined inductively as the smallest set satisfying the following:

1. true and false are propositional formulæ, as is every propositional variable P.
2. If p and q are propositional formulæ then so are $\neg p$, $p \vee q$, $p \wedge q$, $p \Rightarrow q$ and $p \Leftrightarrow q$.

More succinctly, the following is a BNF equation for propositional formulæ.

$$p, q \ ::= \ \mathsf{true} \mid \mathsf{false} \mid P \mid \neg p \mid p \vee q \mid p \wedge q \mid p \Rightarrow q \mid p \Leftrightarrow q$$

Here, P is taken to range over the set of propositional variables.

Exercise 8.6 (Solution on page 441)

Give an inductive definition of the set of formulæ of predicate logic.

BNF notation was invented in 1959 by John Backus (and later simplified by Peter Naur) to define the syntax of the ALGOL programming language. It then became a common feature of the appendix to programming language reference books. This is due to the fact that the set of programs which can be written in a given programming language can be defined inductively from the constructs of the language.

Example 8.6

The following BNF equation describes a very simple programming language.

$$p \ ::= \ x\texttt{:=}e \mid p_1\texttt{;}p_2 \mid \texttt{if } b \texttt{ then } p_1 \texttt{ else } p_2 \mid \texttt{while } b \texttt{ do } p$$

For readability, this is typically rendered in list fashion as follows:

$$\begin{array}{lll} p & ::= & x\texttt{:=}e \\ & \mid & p_1\texttt{;}p_2 \\ & \mid & \texttt{if } b \texttt{ then } p_1 \texttt{ else } p_2 \\ & \mid & \texttt{while } b \texttt{ do } p \end{array}$$

In the above, x is taken to range over program variables; and e and b range over integer expressions and Boolean expressions, respectively, which themselves will similarly be defined inductively. Thus a program in this programming language is either

- an assignment statement "$x\texttt{:=}e$" which evaluates the integer expression e and assigns this value to the variable x; or

- the sequential composition "$p_1\,;p_2$" of two (smaller) programs p_1 and p_2, which first executes the program p_1, and then executes the program p_2 if and when program p_1 has terminated; or
- a conditional statement "`if` b `then` p_1 `else` p_2" involving a Boolean test b and two (smaller) programs p_1 and p_2, which first evaluates the Boolean expression b, and then either executes the program p_1 if b evaluated to true, or executes the program p_2 if b evaluated to false; or
- a while loop "`while` b `do` p" involving a Boolean test b and a (smaller) program p, which repeatedly executes the program p for as long as the Boolean test b is true; that is, it first evaluates the Boolean expression b, and then either terminates if b evaluated to false, or executes the program p and repeats itself (starting with re-evaluating the Boolean expression b) if b evaluated to true.

We shall include one further minor – yet essential – piece of syntax in this language: we will allow ourselves to add braces around any program, thus writing $\{p\}$, in order to avoid ambiguity. This is illustrated in the following example.

The following is a program in this language for computing the sum of the first n positive integers: $s = 1 + 2 + 3 + \cdots + n$.

```
i := 0;
s := 0;
while i<n do
  { i := i+1;
    s := s+i }
```

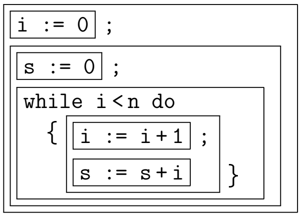
This 5-line program consists of two smaller programs combined with the sequential composition symbol:

```
i := 0 ;
s := 0;
while i<n do
  { i := i+1;
    s := s+i }
```

The first of these programs is a simple assignment statement, while the second program is itself built up from two even smaller programs combined with the sequential composition symbol:

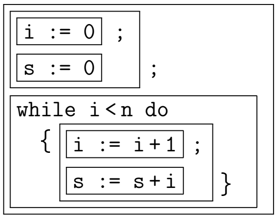
```
s := 0 ;
while i<n do
  { i := i+1;
    s := s+i }
```

Again, the first of these programs is a simple assignment statement, while the second program is a while loop, the body of which is a program consisting of two simple assignment statements combined with the sequential composition symbol. The whole program thus breaks down as follows:



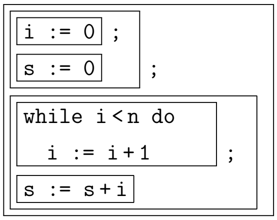
```
i := 0 ;
s := 0 ;
while i<n do
   { i := i+1 ;
     s := s+i  }
```

It is possible to interpret this program differently, namely as two programs combined with the sequential composition symbol, the first being itself two simple assignment statements composed together sequentially, and the second being the while loop. The break down would then look as follows.



```
i := 0 ;
s := 0     ;
while i<n do
  { i := i+1 ;
    s := s+i  }
```

This particular ambiguity is harmless. However, the potential for dangerous ambiguity is why the program includes braces around the body of the while loop. Without these, it would be possible (and moreover likely) that the program would be interpreted wrongly as follows.



```
i := 0 ;
s := 0     ;
while i<n do
  i := i+1     ;
s := s+i
```

This program – or rather this interpretation of the program – would return the incorrect result $s = n$, as the while loop would do nothing but increment the counter i until it reached this value.

8.4 Inductively-Defined Data Types

Most data types used in computer programming languages are inductively defined, either by the compiler (the integers, for example) or by the programmer. For example, a list of natural numbers can be defined by the following BNF equation.

$$L \ ::= \ [\,] \ \mid \ n : L$$

In this definition, n ranges over natural numbers, and the colon symbol ":" represents the operation of adding an element to the front of a list. Thus, a list is either the ***empty list*** $[\,]$ (the list containing no items), or a list obtained by adding a natural number n to the head of a (smaller) list L. For example, the list $[1, 2, 3]$ is built up inductively starting from $[\,]$ as $1 : 2 : 3 : [\,]$. For clarity this could be written using parentheses as $1 : (2 : (3 : [\,]))$.

Of course, we could choose any other type of data to form a list over; e.g., a list of names is defined as above but by letting n range over names rather than numbers.

As a further example, the binary tree is a widely used data structure, and can be defined inductively as follows.

Example 8.7

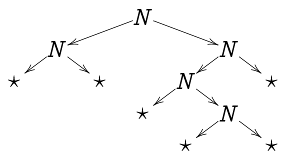
We may inductively define ***binary trees*** using the following BNF equation.

$$t \ ::= \ \star \mid N(t_1, t_2)$$

That is, a tree is either a ***leaf*** $\star$ or an ***internal node*** $N(t_1, t_2)$ with two subtrees t_1 and t_2. For example, the tree

$$N(N(\star, \star), N(N(\star, N(\star, \star)), \star))$$

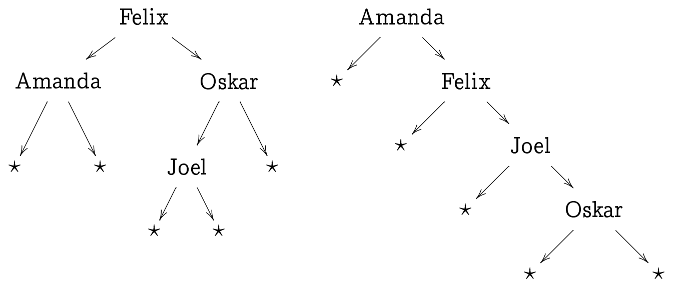
may be represented by the picture shown.



This binary tree definition only provides the *structure* of the data structure, but you typically want to store data in data structures. For example, a ***dictionary*** might be represented by a binary tree with names stored in the (internal) nodes, with the intention that all names stored in the left subtree precede (alphabetically) the name stored in the parent node, and all names stored in the right subtree follow the name stored in the parent node. For example, valid dictionaries for storing the list of names

{Joel, Felix, Oskar, Amanda}

may be given by either of the following trees:



Exercise 8.7 (Solution on page 441)

Give an inductive definition for the the dictionary data structure outlined above. Note that the data structure would only define the syntactic structure; the fact that the names are stored in proper lexicographic order is a semantic issue which will not be reflected in the definition.

8.5 Inductively-Defined Functions

We can exploit the inductive definition of a set to provide convenient definitions for functions on that set. The function is defined by specifying its values on the basic elements of the set, and then specifying its values on the inductively-defined elements in terms of its previously-defined values.

For example, an infinite sequence

$$a_0, a_1, a_2, a_3, a_4, a_5, \ldots$$

is provided by a function whose domain is $\mathbb{N}$, and can often be defined by specifying the initial value a_0 and each subsequent value a_n in terms of the values a_k for $k < n$.

Example 8.8

The ***factorial function*** $n!$ is defined to be the product of the integers from 1 to n:

$$n! = 1 \times 2 \times 3 \times \cdots \times n.$$

More formally, it can be defined inductively as follows.

$$0! = 1; \quad \textit{and}$$

$$n! = n \times (n-1)! \quad \textit{(for } n > 0\textit{)}.$$

Thus, for example,

$$\begin{aligned}5! &= 5 \times 4! \\ &= 5 \times (4 \times 3!) \\ &= 5 \times 4 \times (3 \times 2!) \\ &= 5 \times 4 \times 3 \times (2 \times 1!) \\ &= 5 \times 4 \times 3 \times 2 \times (1 \times 0!) \\ &= 5 \times 4 \times 3 \times 2 \times 1 \times 1 \\ &= 120\end{aligned}$$

Exercise 8.8 (Solution on page 442)

Compute the first few values of the sequence s_n defined inductively by:

$$\begin{aligned}s_0 &= 0 \\ s_n &= s_{n-1} + 2n - 1\end{aligned}$$

Can you recognise this sequence as a function of n?

Example 8.9

The ***harmonic numbers*** H_n are informally defined by

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

and can be defined inductively as follows.

$$\begin{aligned}H_0 &= 0; \quad \textit{and} \\ H_n &= H_{n-1} + \frac{1}{n} \quad \textit{(for } n > 0\textit{)}.\end{aligned}$$

Exercise 8.9 (Solution on page 442)

Compute the harmonic number H_6 from its inductive definition.

Example 8.10

The ***Fibonacci numbers*** are defined inductively as follows.

$$\begin{aligned}f_0 &= 0; \\ f_1 &= 1; \quad \textit{and} \\ f_n &= f_{n-1} + f_{n-2} \quad \textit{(for } n > 1\textit{)}.\end{aligned}$$

That is, each number in this sequence is obtained by adding together the previous two numbers in the sequence. The first few Fibonacci numbers are

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots.$$

This sequence derives its name from the Italian mathematician Leonardo of Pisa, more commonly known by his nickname Fibonacci. Fibonacci was instrumental in spreading the use of the modern Hindu-Arabic numeral system to Europe, as an alternative to Roman numerals, through his book on arithmetic ***Liber Abaci*** *(The Book of Calculation)*, which was published in the early 13th century. The Fibonacci numbers appear in the solution of the following problem posed in this book.

Exercise 8.10 (Solution on page 442)

Suppose you have a pair of new-born rabbits at the start of month 1, and that each pair of rabbits produces a new pair of rabbits after 2 months and each month thereafter. How many pairs of rabbits will you have at the start of the nth month? (Work out the first few months and look for a pattern.)

It is worth looking more carefully at the above inductive definitions of sequences. As the natural numbers $\mathbb{N}$ are defined inductively in terms of zero 0 and the ***successor function*** $s(n) = n{+}1$, functions over them are naturally defined inductively. The above sequences are simple examples, but induction can be used to define more complicated functions than just sequences.

Example 8.11

By resorting to the inductive definition of the natural numbers

$$n \ ::= \ 0 \mid s(n).$$

as given on page 207, we can inductively define the function

$$\mathsf{add} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

which adds two numbers as follows:

$$\begin{aligned} \mathsf{add}(m, 0) &= m; \quad \textit{and} \\ \mathsf{add}(m, s(n)) &= s(\mathsf{add}(m, n)). \end{aligned}$$

The first clause merely states that $m{+}0 = m$; and the second, inductive, clause is the precise way of writing what we would more naturally write as:

$$\mathsf{add}(m, n{+}1) = \mathsf{add}(m, n) + 1.$$

Thus, for example,

$$\begin{aligned}\mathrm{add}(3,2) &= \mathrm{add}(3,1) + 1\\ &= \mathrm{add}(3,0) + 1 + 1\\ &= 3 + 1 + 1\\ &= 5.\end{aligned}$$

Exercise 8.11 (Solution on page 443)

Give an inductive definition of the function

$$\mathrm{mult} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

which multiplies two numbers, in terms of zero and the successor function, as well as the function add defined above.

We can, of course, define functions inductively over any inductively-defined set. The inductive function definitions will naturally follow the structure of the inductive definitions of the domain.

Example 8.12

The length of a word $w \in A^*$ can be defined inductively as follows.

$$\begin{aligned}\mathit{length}(\varepsilon) &= 0\\ \mathit{length}(aw) &= 1 + \mathit{length}(w) \quad \textit{(for } a \in A\textit{)}.\end{aligned}$$

The length of a list of natural numbers can be defined inductively as follows.

$$\begin{aligned}\mathit{length}([\,]) &= 0\\ \mathit{length}(n : L) &= 1 + \mathit{length}(L) \quad \textit{(for } n \in \mathbb{N}\textit{)}.\end{aligned}$$

The height of a binary tree can be defined inductively as follows.

$$\begin{aligned}\mathrm{height}(\star) &= 0\\ \mathrm{height}(N(t_1,t_2)) &= 1 + \max(\mathrm{height}(t_1), \mathrm{height}(t_2)).\end{aligned}$$

Exercise 8.12 (Solution on page 443)

Give an inductive definition of the function $\mathit{sum}(L)$ which computes the sum of a list L of numbers. Use it to verify that $\mathit{sum}([6,2,5]) = 13$.

Exercise 8.13 (Solution on page 443)

The ***append*** function $L_1 +\!\!+\, L_2$ joins two lists L_1 and L_2 together. For example, $[1,2] +\!\!+\, [3,5,7] = [1,2,3,5,7]$. Give an inductive definition of the append function.

Exercise 8.14 (Solution on page 443)

Referring to the inductive definition for formulæ of predicate logic given for Exercise 8.6 (page 208), give an inductive definition for a function which takes a formula of predicate logic and returns the set of variables which appear free in that formula.

8.6 Recursive Functions

In each of the functions defined in the previous section, the value of the function on a given argument is defined either directly, or in terms of its values on smaller arguments. In particular, for functions defined over $\mathbb{N}$ the value of the function on the argument 0 is defined directly, as there are no natural numbers $k<0$.

Such inductively-defined functions are examples of ***recursive functions***, which merely means that the value of a function applied to a given argument is expressed in terms of the value of that function applied to other – not necessarily smaller – arguments. Such definitions may not be well-founded, though. For example, it would not make sense to define a function by $f(n) = f(n+1)+1$; in this case, we'd be forever lost trying to compute $f(0) = f(1)+1 = f(2)+2 = f(3)+3 = \cdots$.

Example 8.14

McCarthy's 91-function $f : \mathbb{N} \to \mathbb{N}$ is defined as follows.

$$f(n) \;=\; \begin{cases} n-10, & \text{if } n > 100; \\ f(f(n+11)), & \text{if } n \leq 100. \end{cases}$$

This function is recursively defined, but not inductively defined. Because of this, it is difficult even to see that this definition is well-founded – that is, that it even defines a value for each argument. In actual fact, $f(n) = 91$ for each $n \leq 100$, and $f(n) = n-10$ for each $n > 100$.

Exercise 8.15 (Solution on page 443)

Prove that McCarthy's 91-function does indeed satisfy $f(n) = 91$ for each $n \leq 100$, and $f(n) = n-10$ for each $n > 100$.

Example 8.15

Consider the following function $f : \mathbb{N} \to \mathbb{N}$.

$$f(n) \;=\; \begin{cases} 1, & \text{if } n \leq 1; \\ f(n/2), & \text{if } n > 1 \text{ even}; \\ f(3n{+}1), & \text{if } n > 1 \text{ odd}. \end{cases}$$

We can attempt to calculate the first few values of f:

$$\begin{aligned}
&f(0) = 1 \\
&f(1) = 1 \\
&f(2) = f(1) = 1 \\
&f(3) = f(10) = f(5) = f(16) = f(8) = f(4) = f(2) = f(1) = 1 \\
&f(4) = f(2) = f(1) = 1 \\
&f(5) = f(16) = f(8) = f(4) = f(2) = f(1) = 1 \\
&f(6) = f(3) = f(10) = f(5) = f(16) = f(8) = f(4) = f(2) = f(1) = 1 \\
&f(7) = f(22) = f(11) = f(34) = f(17) = f(52) \\
&\quad = f(26) = f(13) = f(40) = f(20) = f(10) \\
&\quad = f(5) = f(16) = f(8) = f(4) = f(2) = f(1) = 1 \\
&f(8) = f(4) = f(2) = f(1) = 1
\end{aligned}$$

We quickly realise that the value of the function must be 1 – if it has a value: the only value it could have on some input n is

$$f(n) \;=\; \cdots \;=\; f(1) \;=\; 1.$$

Indeed, this function seems to be well-defined: we don't seem to get into any cycles like

$$f(n) \;=\; \cdots \;=\; f(n);$$

and we always seem eventually to "bottom out" at $f(1) = 1$, although the route to this is rather chaotic: it took 6 unrollings of the function definition to compute $f(5)$, 9 unrollings to compute $f(6)$, and 17 unrollings to compute $f(7)$. It takes 11 unrollings to compute $f(26)$ (as can be seen in the calculation of $f(7)$ above), but it takes no fewer than 112 unrollings to compute $f(27)$, including computing $f(9232)$ along the way which itself requires only 35 unrollings.

It is unknown whether or not this function is in fact well defined, that is, that every sequence $n,\ f(n),\ f^2(n),\ f^3(n),\ \ldots$ eventually arrives at 1, although it has been confirmed for all numbers up to

$$n \;=\; 5.764 \times 10^{18} \;=\; 5{,}764{,}000{,}000{,}000{,}000{,}000.$$

The ***Collatz conjecture*** is the unproven claim that this sequence *does* converge to 1 regardless of the starting value n.

8.7 Recursive Procedures

As the data manipulated by computer programs is typically defined inductively, it should come as no surprise that programs typically manipulate this data recursively. That is, programs are written to run on some input data by recursively calling themselves to run on (generally smaller) input – unless the input data is so trivial that the program can immediately solve the problem at hand.

Example 8.16 Insertion Sort

Consider the problem of sorting a list of integers into increasing order. One method for doing this, called ***Insertion Sort***, works as follows:

1. If the list only has only one element in it, then there is nothing to do: the list is clearly already sorted.
2. Otherwise, put the top card to one side and sort the remaining cards.
3. Insert the reserved card into the correct position in the sorted list.

This breaks the problem of sorting a list of numbers down to that of sorting a smaller list. But the trick is that this procedure is applied *recursively* in Step 2: the smaller list is itself sorted by the same procedure of putting one card to the side and (recursively) sorting the remaining cards – again with the above procedure – before inserting the reserved card into the resulting sorted list.

This procedure is based on the following function defined inductively over lists of numbers:

$$\begin{aligned} isort\ [\,] &= [\,] \\ isort\ (a : L) &= (insert\, a)\ (isort\ \ L) \end{aligned}$$

The definition of the auxiliary function $(insert\, a)$, which inserts the number a into a sorted list, is left as an exercise.

Exercise 8.16 (Solution on page 444)

Define the function $(insert\, a)$ inductively over (sorted) lists of numbers. Your definition should look as follows:

$$\begin{aligned} (insert\, a)\ [\,] &= \cdots \\ (insert\, a)\ (b : L) &= \cdots\ (insert\, a)\ L\ \cdots \end{aligned}$$

You can use the insertion sort procedure to sort a deck of 52 cards into some fixed order, say Ace through King, with all of the Clubs first, followed by the Diamonds, then Hearts, and finally the Spades. To sort the cards, you put the top one down onto a table and sort the remaining 51 cards; to do this, you put the top one down onto the table and sort the remaining 50 cards; continuing in this way, you will eventually find yourself with one card in your hand and 51 cards on the table, which you pick up one-by-one and insert into the correct place into the cards you are holding in your hand.

By doing this, the essence of recursion is hidden; the procedure could simply start with all 52 cards on the table, and picking them up and inserting them one-by-one into your hand, as many bridge players are accustomed to doing. The following example, however, gives a good example of the power of recursion in providing a sorting procedure which works much faster in practice than insertion sort.

Example 8.17 Merge Sort

Another method for sorting a list of numbers, called ***Merge Sort***, works as follows:

1. If the list only has only one element in it, then there is nothing to do: the list is clearly already sorted.
2. Otherwise, divide the list into two equal-sized lists (plus-or-minus one number, if the list consists of an odd number of integers).
3. Sort each of the two shorter lists.
4. Merge the two sorted lists together to produce the desired sorted list.

This breaks the problem of sorting a list of numbers down to that of sorting two smaller lists. But the trick is that this procedure is applied *recursively* in Step 2: the two half-sized lists are each sorted by the same procedure of dividing them into equal-sized lists and (recursively) sorting them – again with the above procedure – before merging them together.

This procedure can be elegantly demonstrated by having a group of people sort a deck of cards. Everyone in the group is to carry out the following procedure if they are handed a pile of cards:

1. If there is only one card in the pile that they are handed, then hand the pile right back to the person who gave it to you.
2. Otherwise, split the pile into two equal-sized piles and pass these smaller piles to two other people who are not holding any cards.
3. Take each of the two piles back when they are handed back to you. You will discover that – as if by magic – these two piles are each sorted.
4. Merge these two sorted piles into one sorted pile, and hand this sorted pile back to the person who gave it to you.

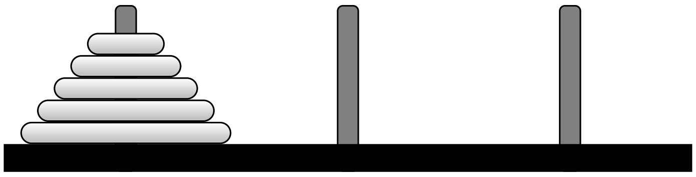


Figure 8.1: Towers of Hanoi with five disks.

Exercise 8.17 (Solution on page 444)

Figure 8.1 depicts the puzzle of the ***Towers of Hanoi*** in which we have three pegs and a number of discs of varying diameter; each disc has a hole in its centre allowing it to be positioned on the pegs. Starting with all of the discs on the first peg in increasing size with the largest on the bottom and the smallest on the top, the puzzle is to move all of the discs to a different peg by moving discs one at a time from peg to peg without ever placing any disc on top of a smaller disc.

Describe a recursive procedure for solving this puzzle. How many individual disc moves would your procedure take on the five-disc puzzle in Figure 8.1?

8.8 Additional Exercises

1. Consider the two quotes given at the start of this chapter and the next chapter. Only one of these properly underlies the principle of inductive definitions. Why is this? (Hint: Consider what the base case may be in each quote.)
2. Consider the set $X \subseteq \mathbb{N}$ defined as follows.

   (a) $0 \in X$.
   (b) if $n \in X$ then $(n+3) \in X$ and $(n+7) \in X$.
   (c) Nothing is in X unless its membership can be established from the above.

   Give three elements of $\mathbb{N}$ which are elements of X, and three elements of $\mathbb{N}$ which are not elements of X, explaining for each one why it is or is not an element.

   Can you give a complete description of the set X?

3. Describe the set P defined as the smallest set satisfying the following:

   (a) $\{\varepsilon, 0, 1\} \subseteq P$.
   (b) if $w \in P$ then $\{0w0, 1w1\} \subseteq P$.

   Give three elements of $\{0,1\}^*$ which are elements of P, and three elements of $\{0,1\}^*$ which are not elements of P, explaining for each one why it is or is not an element.

4. Give an inductive definition of the function *nodecount*(t) which computes the number of internal nodes in the binary tree t, where the definition of a binary tree is as given in Example 8.7. Us this function to verify that

   $$\mathit{nodecount}\Big(N(N(\star,\star), N(N(\star, N(\star,\star)),\star))\Big) \;=\; 5.$$

5. Give a BNF equation for (a fragment of) your favourite programming language.

6. Given an inductive definition of the function *listnames*(d) which takes a dictionary of names d, as defined by Exercise 8.7, and produces a list of names in alphabetic order (assuming the names are properly arranged alphabetically in the dictionary).

7. Give an inductive definition for a function which takes a formula of predicate logic and returns an equivalent formula in which negation symbols appear only applied to predicates.

8. Give an inductive definition of the function *rev* which takes a list and returns its reverse. Thus, for example, $\mathit{rev}(\,[1,2,4]\,) \;=\; [4,2,1]$.

   Use your definition to compute $\mathit{rev}(\,[1,2,4]\,)$.

9. Male bees hatch from unfertilised eggs, and so have a mother but no father. Female bees hatch from fertilised eggs, and so have both a mother and a father. The family tree of a male bee can be seen in Figure 8.2 How many ancestors does a male bee have in the tenth generation back? How many of these ancestors are male?

10. Give an inductive definition of the function *msort* upon which merge sort is based. You will want to define auxiliary functions *split* which splits a list into two equal-size lists, and *merge* which merges two sorted lists into one list.

11. Ackermann's Function is defined inductively as follows. For $n \geq 0$,

    $$A(0,n) \;=\; n+1;$$

    and for $m, n \geq 1$,

    $$\begin{aligned} A(m,0) &= A(m{-}1,1) \qquad \textit{and} \\ A(m,n) &= A(m{-}1, A(m,n{-}1)). \end{aligned}$$

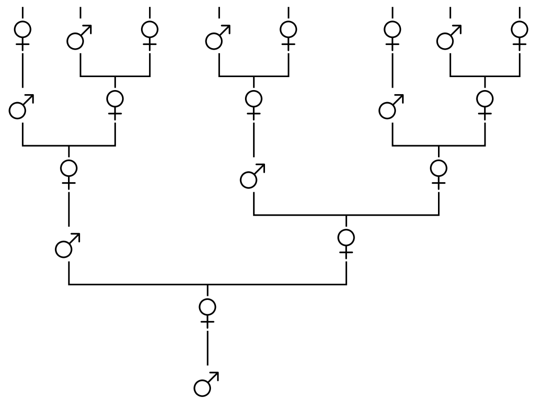


Figure 8.2: A family tree of male (♂) and female (♀) bees.

This is an extremely fast growing function. For example, that value of $A(4, 2)$ has $19, 729$ decimal digits; and the value of $A(4, 3)$ is already well beyond astronomical.

(a) Work out the first few values of $A(1, n)$ to convince yourself that $A(1, n) = n+2$.

(b) Work out the first few values of $A(2, n)$ to convince yourself that $A(2, n) = 2n+3$.

(c) Work out the first few values of $A(3, n)$ to convince yourself that $A(3, n) = 2^{n+3} - 3$.

(d) Work out the value of $A(4, 1)$.