CHAPTER 6

Functions

6.1 Functions and computing

If you have already studied functions in mathematics, it is likely that what you were studying was a type of function known as a 'real-valued function of a real variable'. If this is the case, then you probably think of a function as 'something that has a mathematical formula' such as $x^2 - 2x + 3$, or 'something you can draw the graph of' using x and y axes.

In computing (and in many areas of mathematics, for that matter), we take a different approach to functions, which turns out to be more useful. Our definition of a function is considerably more general, and in most cases it will not be possible to draw graphs of the functions we will be studying. The way we think about functions will also be somewhat different. The functions used in programming in high-level languages (both the functions available in libraries associated with those languages and the function subprograms that you write yourself) are usually functions in this more general sense (with some qualifications, as we will see later). Functions of a real variable will be needed in Chapter 13 when we study the time complexity of algorithms.

Definition Let X and Y be sets. A *function* from X to Y is a rule that assigns to each element of X exactly one element of Y.

We will generally use lower case letters such as f, g and h to denote functions. Greek letters such as ϕ (phi) are also commonly used. If f is a function from X to Y, we indicate this by writing $f:X \to Y$. X is called the *domain* of f, and Y is the *codomain*.

If $f:X \to Y$ is a function with domain X and codomain Y, and if x is any element of X, then according to the definition there is exactly one element of Y assigned to x. That element is called the *image* of x, and is written $f(x)$.

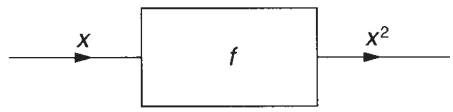
For our first example, we will take a function you have probably seen before, and see how it can be viewed in the light of the definition.

Let the function f be defined as follows:

$$f:\mathbf{R} \to \mathbf{R},\ f(x) = x^2$$

This function assigns to each element of its domain **R** (that is, to each real number) an element of its codomain (which is also **R**). The rule that specifies which element to assign is $f(x) = x^2$, that is, the function f assigns to each real number the square of that number. If we pick any element of the domain, we can find the image of that element; for example, $f(5) = 25$, $f(-3) = 9$, $f(\sqrt{2}) = 2$. We see that every element of the domain **R** has a unique image, and each such image is an element of the codomain **R**. This ensures that f is a function, according to the definition.

A helpful way of thinking about this function is to imagine a machine that accepts input and produces output:



The function f is a 'squaring machine'. We may feed in any element of the domain (that is, any real number) as input, and the machine will square it and output the result. We can think of the symbol f as standing for the *process* of squaring a number. The domain of f is the set of valid inputs, and the codomain is a set to which all of the possible outputs belong.

It is important to be clear about what the notation means – $f(x)$ denotes the element of the codomain that is the image of an element x of the domain, whereas f by itself is the name of the function, and represents the process that the function carries out. We could have written the rule for the function as $f(y) = y^2$ and it would still be the *same* function, because it carries out the same squaring process. In the terminology of predicate logic, the variable x in the rule for the function is a *bound* variable, because the rule really means '$\forall x[f(x) = x^2]$'.

We want to make one more observation before we leave this example. It concerns the codomain **R**, to which all the images $f(x)$ belong. Notice that some elements of the codomain are not images of anything. For example, $-1 \in \mathbf{R}$, but -1 is not the image of any element of the domain, because the square of a real number cannot be negative. There is nothing in the definition of a function to say that all of the codomain has to be 'used'; the set of images of elements of the domain is a subset of the codomain, and it can be a proper subset.

If $f: X \to Y$ is any function, the set of images $\{y \in Y : y = f(x)$ for some $x \in X\}$ is called the *range* of f. The range of a function is a subset of the codomain.

In the example, the range of f is the set of non-negative real numbers: $\{y: y \geq 0\}$.

The function we have just been looking at is an example of a real-valued function of the kind you might study in a calculus course. In the next few examples we want to consider other types of functions, especially those that might occur in problems related to computing.

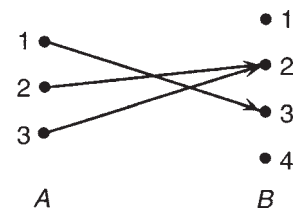
For our second example of a function, let $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4\}$. Let f be the function defined in the following way:

$$f: A \to B,\ f(1) = 3,\ f(2) = 2,\ f(3) = 2$$

This is a perfectly well defined function (although it is probably not a very useful one). It doesn't have a 'formula' like x^2, but that doesn't matter. All that matters is that *every element of the domain has exactly one image, and that image is an element of the codomain*. The elements of the domain A are 1, 2 and 3, their images are 3, 2 and 2 respectively, and these images are all in the codomain B. That is all that is needed in order for f to be a function. We see that the elements of B that occur as images are 2 and 3, so the range of f is {2, 3}.

While it would be possible to draw a graph of this function in the conventional sense, it would not be particularly useful to do so. A more appropriate way of depicting a function like this is by means of an arrow diagram. The arrow diagram for f is shown in Figure 6.1. Notice that exactly one arrow emerges from each element of the domain. This must always be the case if the diagram is to represent a function.

Figure 6.1



For our third example, let C be the set of all ASCII characters. Recall that each ASCII character has a unique character code in the range of integers from 0 to 127, and conversely that each integer in that range is the ASCII code of exactly one character. We can define the following functions:

$$ord : C \to \{0,1,2,\ldots,127\},\ ord(c) = \text{ASCII code of } c$$

$$chr : \{0,1,2,\ldots,127\} \to C, chr(n) = \text{character with ASCII code } n$$

(The notation *ord* and *chr* for these functions is used in Pascal.)

In order to find the image of an element of the domain of either of these functions, we would need to refer to a table of ASCII characters. For example, by looking up an ASCII table we find that *ord*('A') = 65, *ord*('a') = 97, and *ord*('*') = 42. It follows immediately that *chr*(65) = 'A', *chr*(97) = 'a', and *chr*(42) = '*'.

Each of the functions *ord* and *chr* has for its range the entire codomain of the function.

Example 6.1.1 Let X = {finite non-empty strings of bits} and Y = {0, 1, 2, 3, ...}. Determine whether the following functions are well defined:

(a) $f: X \to Y$, $f(s)$ = number of ones in s

(b) $g: X \to Y$, $g(s)$ = first bit of s

(c) $h: X \to Y$, $h(s)$ = position in the string of the leftmost zero of s

(d) $j: X \to X$, $j(s)$ string obtained by appending 0 or 1 to s

(e) $k: Y \to X$, $k(n)$ string of n ones

Solution

(a) Given any finite non-empty string of bits, s, it is always possible to find the number of ones in s, and the result is always an element of Y. Therefore f is a function.

(b) The first bit of a finite non-empty string can always be found, and it is either 0 or 1. Since 0 and 1 are both elements of Y, g is a function.

(c) Some bit strings do not contain zeros, so h is not well defined.

(d) The image of an element is not uniquely determined, because either 0 or 1 can be appended, so j is not well defined.

(e) Given any element n of Y, it is possible to write down a string of n ones. However, if $n = 0$ the string will be the empty string, which is not an element of X. Therefore k is not well defined.

We have already seen examples in which the range of the function is not all of the codomain. The next definition refers to the situation where the range *is* the entire codomain.

Definition A function is *onto* if its range is equal to its codomain.

Equivalently, a function is onto if every element of the codomain is the image of at least one element of the domain. (The use of 'onto' as an adjective may seem strange, but the terminology is well established.)

It is possible for two different elements of the domain of a function to have the same image in the codomain. In our first example (the 'squaring' function), 2 and –2 have the same image: $f(2) = 4$ and $f(-2) = 4$. The next definition provides the terminology for the functions for which this does not happen.

Definition A function is *one-to-one* if no two distinct elements of the domain have the same image.

In order to show that a function f is one-to-one, we need to show that if x_1 and x_2 are elements of the domain and $x_1 \neq x_2$, then $f(x_1) \neq f(x_2)$. (In practice, it is usually easier to prove the contrapositive: if $f(x_1) = f(x_2)$, then $x_1 = x_2$.) In order to show that a function is not one-to-one, on the other hand, it is sufficient to find two elements of the domain with the same image (as we did with the 'squaring' function).

Example 6.1.2 For each of the following functions, determine whether the function is onto and whether it is one-to-one:

(a) $f: \mathbf{R} \to \mathbf{R}$, $f(x) = 2x + 1$

(b) The function *ord* defined earlier

(c) The function *chr* defined earlier

(d) The function f in Example 6.1.1(a)

(e) The function g in Example 6.1.1(b)

(f) $\phi: X \to X$, $\phi(s)$ = string obtained by appending 0 to s (with X as defined in Example 6.1.1)

Solution

(a) If $y \in \mathbf{R}$, then we can set $y = 2x + 1$ and solve for x, obtaining $x = (y - 1)/2$. Therefore every element y of the codomain is the image of the element $(y - 1)/2$ of the domain. Thus f is onto.

We show that f is one-to-one by proving the contrapositive of the definition. Suppose $f(x_1) = f(x_2)$. Then:

$$2x_1 + 1 = 2x_2 + 1$$

Hence $$2x_1 = 2x_2$$

so $$x_1 = x_2$$

Therefore f is one-to-one.

(b) Every integer from 0 to 127 is the ASCII code of a character in the ASCII character set, so *ord* is onto. No two characters have the same code, so *ord* is one-to-one.

(c) Every character has an ASCII code, so *chr* is onto. No two codes correspond to the same character, so *chr* is one-to-one.

(d) Recall that the function is $f: X \to Y$, $f(s)$ = number of ones in s.

Is every element of Y the image of something? In other words, if we count the number of ones in a string of bits, could we get any of the numbers 0, 1, 2, ... as the result? The answer is yes, so f is onto.

Is it possible for two different elements to have the same image? In other words, could two *different* strings of bits yield the *same* result when we count the number of ones? The answer is clearly yes (101 and 110 is one of many examples), so f is not one-to-one.

(e) Recall that the function is $g: X \to Y$, $g(s)$ = first bit of s.

Is every element of Y the image of something? No; 2 is not the image of any element of X, so g is not onto.

Can two different elements have the same image? Yes; 00 and 01, for example, so g is not one-to-one.

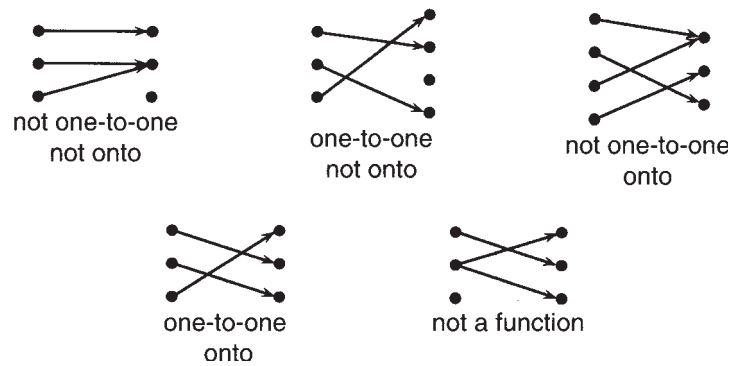
(f) Is every element of X the image of something? No; 01 is not, because it doesn't end in 0. Therefore ϕ is not onto.

Can two different elements have the same image? If so, this would mean that two different bit strings would give the same bit string when 0 is appended to them, which is clearly impossible. Therefore ϕ is one-to-one.

The solution to Example 6.1.2 reinforces an important point made in Chapter 5 when we studied equivalence relations. You have already made a substantial step towards solving a mathematical problem when you have identified just what it is that you need to show. In order to determine whether a function is onto, you need to ask yourself: *Is every element of the codomain the image of something in the domain?* In order to determine whether a function is one-to-one, the question you need to ask is: *Can two different elements of the domain have the same image?* You then need to interpret the question in the context of the particular problem. This is what we were doing in Example 6.1.2.

Sketching an arrow diagram can also help you to decide whether a function is onto and whether it is one-to-one. Some examples of typical arrow diagrams for the various types of functions are shown in Figure 6.2.

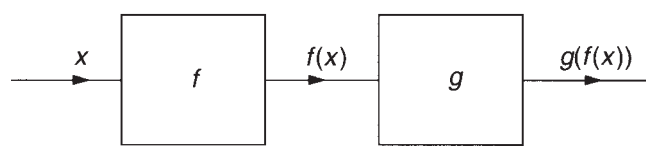
Figure 6.2



The last diagram in Figure 6.2 is not the arrow diagram of a function, because it fails the requirement that every element of the domain must have exactly one image. (Note that the terms 'onto' and 'one-to-one' apply only to functions. If you have determined that what you have been given is not a function, then the question of whether it is onto or one-to-one does not arise.)

6.2 Composite functions and the inverse of a function

Suppose f and g are two functions. If we think of f and g as machines with input and output, we could imagine linking them together so that the output of f becomes the input of g:



This will work only if the output from f belongs to the domain of g. In order to ensure that this is always the case, we will assume that the codomain of f equals the domain of g. Specifically, let A, B and C be arbitrary sets, and let $f: A \to B$ and $g: B \to C$.

We can now think of the combination of the two machines as a single machine with input $x \in A$ and output $g(f(x)) \in C$. This new machine corresponds to a function from A to C, called the *composite* function of f and g.

The formal definition follows.

Definition Let $f: A \to B$ and $g: B \to C$ be functions. The *composite function* of f and g is the function:

$$g \circ f : A \to C, (g \circ f)(x) = g(f(x))$$

Notice that $g \circ f$ needs to be read from right to left: it means *first* apply f, then apply g to the result.

Although real-valued functions are only of minor importance for our purposes, we will use them in our first example because you are likely to be more familiar with them.

Example 6.2.1 Let $f: \mathbf{R} \to \mathbf{R}$, $f(x) = x^2$ and $g: \mathbf{R} \to \mathbf{R}$, $g(x) = 3x - 1$. Find $f \circ g$ and $g \circ f$.

Solution Note firstly that the composite function $f \circ g$ exists because the codomain of g equals the domain of f. Similarly, $g \circ f$ exists because the codomain of f equals the domain of g.

The function $f \circ g$ is found as follows:

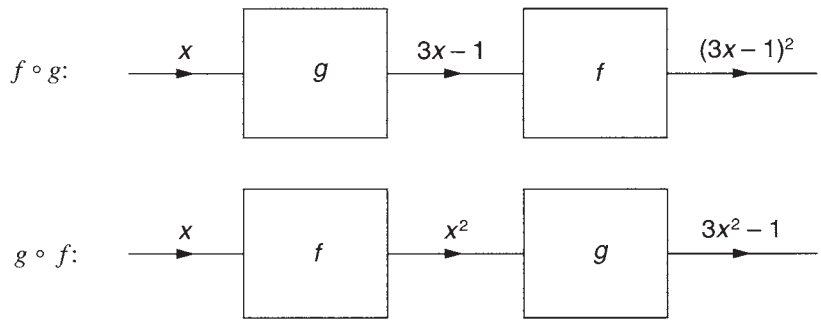
$$\begin{aligned} f \circ g : \mathbf{R} \to \mathbf{R}, (f \circ g)(x) &= f(g(x)) \\ &= f(3x-1) \\ &= (3x-1)^2 \end{aligned}$$

where the last line is obtained by substituting $3x - 1$ in place of x in the formula for $f(x)$.

The function $g \circ f$ is obtained in a similar manner:

$$\begin{aligned} g \circ f : \mathbf{R} \to \mathbf{R}, (g \circ f)(x) &= g(f(x)) \\ &= g(x^2) \\ &= 3x^2 - 1 \end{aligned}$$

The two composite functions in the last example can be depicted in the following way:



Example 6.2.2 Let X be the set of all finite non-empty strings of characters. Let the functions f and g be defined as follows:

$$f: X \to \mathbf{N},\ f(s) = \text{number of characters in } s$$

$$g: X \to X,\ G(s) = \text{string obtained by appending 'a' to } s$$

State whether the following functions exist. For those that do exist, describe the function as simply as possible.

(a) $f \circ f$
(b) $f \circ g$
(c) $g \circ f$
(d) $g \circ g$

Solution

(a) The codomain of f does not equal the domain of f, so $f \circ f$ does not exist.

(b) The codomain of g equals the domain of f, so $f \circ g$ exists. The composite function appends 'a' to a string and counts the number of characters in the resulting string. It can be described in the following way:

$$f \circ g : X \to \mathbf{N}, (f \circ g)(s) = (\text{number of characters in } s) + 1$$

(c) The codomain of f does not equal the domain of g, so $g \circ f$ does not exist.

(d) The codomain of g equals the domain of g, so $g \circ g$ exists. The composite function appends 'a' to a string, then appends 'a' to the resulting string:

$$g \circ g: X \to X,\ (g \circ g)(s) = \text{string obtained by appending 'aa' to } s$$

We now turn to the problem of defining the inverse of a function. In Section 6.1, we defined the following two functions:

$$ord : C \to \{0,1,2,...,127\}, ord(c) = \text{ASCII code of } c$$

$$chr : \{0,1,2,...,127\} \to C, chr(n) = \text{character with ASCII code } n$$

There is a close relationship between *ord* and *chr*. Firstly, the domain of *ord* equals the codomain of *chr*, while the codomain of *ord* equals the domain of *chr*. Secondly, each function 'undoes' or reverses the effect of the other; *ord* converts a character to its corresponding code, while *chr* converts a code to its corresponding character. We express this fact by saying that *ord* and *chr* are *inverses* of each other.

This idea can be made more precise by looking at the composite functions $ord \circ chr$ and $chr \circ ord$. The function $ord \circ chr: \{0,1,2,\ldots,127\} \to \{0,1,2,\ldots,127\}$ converts a number to the corresponding character, and then converts the resulting character back to the original number. The overall effect of $ord \circ chr$ is to leave every number unchanged. Similarly, the function $chr \circ ord: C \to C$ has the overall effect of leaving every character unchanged.

Before we can define the inverse of a function formally, we need another definition.

Definition Let A be a set. The *identity function* on A is the function:

$$i: A \to A,\ i(x) = x$$

The identity function is a 'do nothing' function; it simply maps each element of A to the element itself. It can be thought of as a machine that outputs anything it receives as input.

While it has to be admitted that the identity function on a set is not a very interesting function, it does have some important properties. Firstly, it is one-to-one and onto. Secondly, if f is any function with domain A, and if i denotes the identity function on A, then $f \circ i$ is the same function as f itself. Similarly, if g is a function with codomain A, then $i \circ g$ is the same function as g.

We can now give the formal definition of the inverse of a function.

Definition Let $f: A \to B$ and $g: B \to A$ be functions. If $g \circ f: A \to A$ is the identity function on A, and if $f \circ g: B \to B$ is the identity function on B, then f is the *inverse* of g (and g is the inverse of f).

Not every function has an inverse, as we will see shortly. If a function does have an inverse, it can have only one.

The inverse of a function f is denoted by f^{-1}. It is best to think of this simply as the notation for the inverse of a function as defined above. Don't think of it as 'f to the power of –1', as it is quite different from raising a *number* to the power –1, for example $2^{-1} = \frac{1}{2}$. In particular, we never write $1/f$ for the inverse of f.

The domain of f^{-1} is the codomain of f, and vice versa. We can think of f^{-1} as a machine that 'reverses' f – it takes any valid output of f as its own input, and produces as output the corresponding input of f.

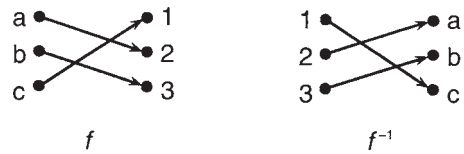
Example 6.2.3 Let f: {a, b, c} → {1, 2, 3}, f(a) = 2, f(b) = 3, f(c) = 1. Find f^{-1}.

Solution The domain of f^{-1} is the codomain of f and vice versa. The rule for f^{-1} is obtained by reversing the rule for f:

$$f^{-1}: \{1,2,3\} \to \{\text{a, b, c}\}, f^{-1}(1) = \text{c}, f^{-1}(2) = \text{a}, f^{-1}(3) = \text{b}$$

The arrow diagrams for the functions f and f^{-1} in Example 6.2.3 are shown in Figure 6.3. The diagram for the inverse is obtained by reversing the arrows; the arrow from c to 1 becomes an arrow from 1 to c, and so on.

Figure 6.3



The arrow diagram gives us a clue to the situations in which a function has no inverse. If f is not onto, the arrow diagram obtained by reversing the arrows will have elements of the domain with no image, and therefore it cannot represent a function. If f is not one-to-one, the reversed diagram will have at least one element with two images, so it cannot represent a function in this case either. This observation is summarised in the following theorem.

Theorem A function f has an inverse if and only if f is onto and one-to-one.

If a function $f: X \to Y$ is onto and one-to-one, then f establishes an exact one-to-one correspondence between the elements of X and the elements of Y. It follows that X and Y must have the same cardinality. We will see a practical consequence of this fact shortly.

Example 6.2.4 Determine which of the following functions have inverses:

(a) $f: \mathbf{R} \to \mathbf{R}$, $f(x) = 2x + 1$

(b) $g: \mathbf{R} \to \mathbf{R}$, $g(x) = x^2$

(c) $h: \{x \in \mathbf{R}: x \geq 0\} \to \{x \in \mathbf{R}: x \geq 0\}$, $h(x) = x^2$

Solution

(a) This function is onto and one-to-one, so f^{-1} exists. To find a formula for the inverse function, write $x = 2f^{-1}(x) + 1$ and solve for $f^{-1}(x)$ to obtain $f^{-1}(x) = (x - 1)/2$. Intuitively, f 'doubles and adds 1', while f^{-1} 'subtracts 1 and divides by 2', which is the reverse process.

As a check on the answer, note that

$$(f \circ f^{-1})(x) = f(f^{-1}(x)) = f\left(\frac{x-1}{2}\right) = 2\left(\frac{x-1}{2}\right)+1 = x,$$

and similarly that $(f^{-1} \circ f)(x) = x$.

(b) This function is neither onto nor one-to-one, so it has no inverse.

(c) Notice that this is not the same function as the one in (b), because both the domains and the codomains are different. The function h is one-to-one, because it is impossible for two non-negative real numbers to have the same square. The function is also onto. The rule for the inverse function is $h^{-1}(x) = \sqrt{x}$.

Here is an example of a practical situation where inverse functions arise. It is often desirable to encrypt a confidential message prior to transmitting it via a possibly insecure channel, in order to ensure that it cannot be read by an unauthorised person. Let X be the set of all possible original messages, and let Y be the set of all encrypted messages. Then the code used to carry out the encryption process can be thought of as a function $f: X \to Y$. In order to ensure that any message can be decoded, there must be an inverse function $f^{-1}: Y \to X$ to carry out the decoding process. This means that any function f that we might consider using to perform the encryption must be one-to-one and onto.

A similar situation arises with software for file compression. The process of compressing a file so that it occupies less space on a disk can be thought of as an encryption process corresponding to a function f. The data in a file is stored on a disk as a finite (and non-empty) string of bits, and we will assume that there is an upper limit N on the number of bits in any file. If we also assume that no string of N bits or fewer can be ruled out as the possible contents of a file, then the domain of f is the set X of all bit strings with N bits or less. The codomain is the set Y of those bit strings that can occur as the contents of a compressed file.

The function f must have an inverse function f^{-1} to perform the process of expanding a compressed file to retrieve the original data. This means that f must be one-to-one and onto, and therefore that X and Y must have the same cardinality. Of course, we would also like the function f to have the property that each compressed file is smaller than the corresponding original file. Can we find such a function?

The answer is no! If each compressed file is smaller than the original file, then Y must contain only files with fewer than N bits. This makes Y a proper subset of X, so Y must contain fewer elements than X. We arrive at the following surprising conclusion: *there is no file compression algorithm that compresses every file*.

It is actually possible to prove an even stronger statement: Any file compression algorithm that makes at least one file smaller must also make at least one file larger.

The file compression utilities in common use can be very effective in compressing most of the files that occur in practice. With any such

program, however, there must always be files that cannot be compressed with that program.

6.3 Functions in programming languages

Most programming languages (including spreadsheet and database languages) include a construct called a 'function' as part of the language. There are typically many functions available in function libraries, and programmers can also write their own functions. How are these functions related to functions in the mathematical sense?

Spreadsheet software typically includes a comprehensive library of functions for manipulating data stored in a spreadsheet. These functions can be interpreted as functions in the sense we have been using in this chapter. Some of the functions available in spreadsheets are mathematical functions in the familiar sense. For example, The function ABS takes a real number x as input, and returns its absolute value $|x|$ as output. We can therefore take the domain and codomain to be the set of real numbers, and write:

$$\text{ABS}: \mathbf{R} \to \mathbf{R}$$

Strictly speaking, we are using **R** here to denote the set of *computer representations* of real numbers, rather than the set of real numbers in the mathematical sense.

Another example of a spreadsheet function is a function to determine whether the content of a cell is a text string. Such a function might be called ISTEXT. ISTEXT can take as input an item of data of any type (character string, number value or logical condition), or (more usually) the address of a cell in the spreadsheet. The output is TRUE if the input data (or the data contained in the cell address) is a text string, and FALSE if it is not. If X denotes the set of all valid items of data and all valid cell addresses, then:

$$\text{ISTEXT}: X \to \{\text{TRUE}, \text{FALSE}\}$$

A third example is provided by a function, LEFT, which takes two arguments: a character string s and a natural number n, and returns as output the string consisting of the first n characters of s; for example, LEFT("Hello", 3) = "Hel". If we want to interpret LEFT as a function in the mathematical sense, we must write the domain as a Cartesian product. If S denotes the set of all character strings, then:

$$\text{LEFT}: S \times \mathbf{N} \to S$$

The term 'function' in spreadsheet programs also embraces constants such as PI, which returns a numerical approximation to π. This 'function' has no arguments (think of a machine that outputs the value of π without needing any input). Somewhat artificially, we can think of PI as a function whose domain contains one (unspecified) element:

$$\text{PI}: \{*\} \to \mathbf{R}, \ \text{PI}(*) = 3.1415926$$

This is really stretching our definition of 'function'!

Programming languages such as Pascal and C, in addition to providing standard libraries of functions, allow you as the programmer to write your own functions. In both Pascal and C, the syntax rules for the language require you to specify the data types of the input and output in the statement that heads the function subprogram. A typical function header in Pascal might look like this:

```
function f(x: char; y: real): integer;
```

In C, the header would look like this:

```
int f(char x, float y)
```

The body of the function follows the header, and includes an algorithm for evaluating the function in terms of the arguments `x` and `y`.

If we treat this as a mathematical function f, then the domain of f is a subset of $C \times \mathbf{R}$, where C denotes the set of all characters available on the machine, and we can take the codomain to be $\mathbf{J}$.

In many programming languages, however, it is possible for the value returned by a function to depend on more than just the arguments (it can depend on the value of a global variable, for example). A function can also produce 'output' in addition to the value it returns (by changing the value of a global variable, for example, or by writing a message to the screen). Because of these 'side-effects', a function can return two different results when it is called twice with the same arguments. In C and some other languages, a function need not return a value at all. For these reasons, a function in a programming language may not be a function in the mathematical sense.

One example of a function with side-effects is a pseudo-random number generator. Pseudo-random number generators are available as standard library functions in many programming languages, and are widely used in simulation software. A typical pseudo-random number generator might produce a different real number in the interval from a to b each time it is called with arguments a and b. At first sight, it might appear to be a function in the mathematical sense:

$$random: \mathbf{R} \times \mathbf{R} \to \mathbf{R},\ random(a, b) = \text{a random number between } a \text{ and } b$$

When we use *random* in a program, we might find, for example, that $random(0,1) = 0.3982473$ on the first call and $random(0,1) = 0.8194702$ on the second call. Since the result is not uniquely determined by the values of the two arguments, we conclude that *random* does not qualify as a function in the mathematical sense of the word.

In this chapter, we have introduced the powerful notion of a function from any set to any other set, and we have seen how functions in this general sense arise in computing contexts. Functions play a central role in mathematics, and they will appear frequently in the chapters that follow.

EXERCISES

1 Determine whether the following functions are well defined. For those that are well defined, state the domain, codomain and range.

(a) $f: \mathbf{R} \to \mathbf{R},\ f(x) = \dfrac{x}{3}$

(b) $g: \mathbf{J} \to \mathbf{J},\ g(x) = \dfrac{x}{3}$

(c) $h: \{1,2,3\} \to \{1,2,3\},\ h(n) = n+1$

(d) $d: \mathbf{N} \to \mathbf{N},\ d(n) =$ number of digits in the decimal representation of n

(e) $\phi: X \to \mathbf{R},\ \phi(x) = 1/x$, where X is the 'maximal' domain (that is, the largest subset of $\mathbf{R}$ such that $1/x$ is well defined when $x \in X$)

(f) $ispositive: \mathbf{R} \to \{\text{T, F}\},\ ipositive(x) = \begin{cases} \text{T if } x \text{ is positive} \\ \text{F if } x \text{ is negative} \end{cases}$

(g) $r: \mathbf{N} \to \mathbf{J},\ r(n) =$ remainder after n is divided by 6

(h) $\psi: S \to S,\ \psi(s) =$ string obtained by removing the last character from s (where S is the set of finite non-null character strings)

2 Determine which of the following functions are one-to-one and which are onto:

(a) $f: S \to S,\ f(s) =$ string obtained by reversing the order of the characters of s (S = {finite non-null character strings})

(b) $g: \mathbf{R} \times \mathbf{R} \to \mathbf{R},\ g((x, y)) = x + y$

(c) $s: \mathbf{N} \to \mathbf{N},\ s(n) = n+1$

(d) h: {English words} → {letters}, $h(w)$ = first letter of word w

(e) $b: \mathbf{N} \to$ {finite non-null bit strings}, $b(n)$ = binary representation of n (written without leading zeros)

(f) $card: \mathcal{P}(A) \to \mathbf{N} \cup \{0\},\ card(X) =$ the cardinality of the set X (where A denotes any finite set)

3 A function f: {1, 2, 3, 4, 5} → {0, 1, 2, 3, 4} is defined by the rule: $f(n)$ is the remainder after $3n$ is divided by 5. Draw an arrow diagram for this function. Hence state whether f is one-to-one and whether f is onto.

4 For the purpose of error detection, numeric codes (such as ID numbers) often include a final 'check digit'.

Suppose a numeric code consists of a string of 9 digits $x_1x_2...x_9$, followed by a final check digit x_{10} defined to be the rightmost decimal digit of $x_1 + 2x_2 + 3x_3 + ... + 9x_9$.

(a) Verify that 2516238674 is a valid code.

(b) Let X be the set of all strings of 9 digits, let Y be the set of all digits, and let $f: X \to Y$ be the function that assigns the correct check digit to each string, for example $f(251623867) = 4$. State, giving reasons, whether f is one-to-one and whether f is onto.

(c) If an error is made in keying in a code, will the check digit always detect it? Explain, with reference to your answer to (b).

5 Let the functions f, g and h be defined as follows:

$$f : \mathbf{R} \to \mathbf{R}, f(x) = 4x - 3$$

$$g : \mathbf{R} \to \mathbf{R}, g(x) = x^2 + 1$$

$$h : \mathbf{R} \to \mathbf{R}, h(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

Find rules for the following functions with domain and codomain **R**:

(a) $f \circ f$ (b) $f \circ g$

(c) $g \circ f$ (d) $f \circ h$

(e) $h \circ f$ (f) $g \circ h$

(g) $h \circ g$

6 Find the inverse function of each of the following functions, or explain why no inverse exists:

(a) $f: \mathbf{R} \to \mathbf{R}$, $f(x) = 3x + 2$

(b) $abs: \mathbf{R} \to \mathbf{R}$, $abs(x) = |x|$

(c) $g: \mathbf{N} \to \mathbf{N}$, $g(n) = \begin{cases} n+1 \text{ if } n \text{ is odd} \\ n-1 \text{ if } n \text{ is even} \end{cases}$

(d) $h: S \to S$, $h(s) =$ the string obtained by moving the last character to the beginning of the string, for example h('abcd') = 'dabc' (where S is the set of finite non-null character strings)

7 Let X be the set of all names of students in a database maintained by a university. Assume that no two students have the same name. Let Y be the set of ID numbers of the students. The functions f and g are defined as follows:

$f: X \to Y$, $f(x) =$ ID number of student with name x

$g: Y \to \mathbf{N}$, $g(y) =$ age (in years) of student with ID number y

(a) Describe the functions $g \circ f$ and f^{-1}.

(b) Explain why g^{-1} does not exist.

8 Let S be the set of all finite non-null strings of characters. Let $upr: S \to S$ be the function that converts all lower-case letters to upper-case (and leaves all other characters unchanged). Similarly, let $lwr: S \to S$ be the function that converts all upper-case letters to lower-case.

(a) Evaluate upr('Barbara Hill') and $(lwr \circ upr)$('Barbara Hill').

(b) Are upr and lwr inverses of each other? Explain.

9 Let $X = \{1, 2, 3, ..., 20\}$, and let Y be the set of non-null strings of up to 20 characters. Let f be defined as follows:

$f: X \to Y$, $f(x) =$ English word for x (in lower-case letters)

For example, $f(1) =$ 'one'.

(a) State, giving reasons, whether:

(i) f is one-to-one;

(ii) f is onto;

(iii) f^{-1} exists.

(b) If $g: Y \to X$, $g(s) =$ number of characters in s, evaluate the following expressions, where possible:

(i) $(g \circ f)(7)$

(ii) $(g \circ f)$('seven')

(iii) $(f \circ g)(7)$

(iv) $(f \circ g)$('seven')

10 Let $f: \mathbf{N} \to \mathbf{N}$, $f(n) =$ digital root of n. (Digital roots were defined in Chapter 1, Exercise 8.)

(a) State, with reasons, whether f is one-to-one and whether f is onto.

(b) Does f^{-1} exist? If so, describe it. If not, give a reason.

(c) Does $f \circ f$ exist? If so, describe it. If not, give a reason.

11 Let A, B, C and D be sets, and let $f: A \to B$, $g: B \to C$ and $h: C \to D$ be functions. Prove that $(h \circ g) \circ f = h \circ (g \circ f)$.

12 Prove that a function cannot have more than one inverse. (Hint: Let $f: A \to B$ be a function, and assume f has two inverses, g_1 and g_2. Deduce a contradiction by evaluating $g_1 \circ f \circ g_2$ in two different ways.)