2.8 THE DESIGNATION OF FUNCTIONS AND THE LAMBDA NOTATION

We have used the conventional means of designating functions in writing defining expressions such as

$$f(x) = x^2$$

to indicate that function which maps every value of the variable x into the number x^2. There are, however, some disquieting aspects of this notation. We cannot take the "=" sign as meaning that the right-hand side is a complete description of the function f, that it is in a sense equivalent to the map $x \to x^2$, because some essential information is missing from it. To see this, suppose that we write what should be an equivalent designation if f is defined on all the integers:

$$f(x + 1) = x^2 + 2x + 1.$$

The right-hand side, viewed alone, does not tell us whether it is intended to designate the function

$$x + 1 \to x^2 + 2x + 1,$$

which is a rewriting of $x \to x^2$, or perhaps,

$$x \to x^2 + 2x + 1,$$

which is a quite different function. The missing information, the argument of f, appears only on the left-hand side.

Or, again, if we write

$$f(x) = 3x + y,$$

where y is some arbitrary quantity whose value is to be assigned (a parameter), we cannot tell from the right-hand side $3x + y$ (which is an example in mathematics of a "form") whether the intent is that x or that y be the independent variable. Also, if we write $g(s, r) = 2r + 3s$ and ask for the value of g at $(4, 1)$, it is not clear, without further convention, whether the intent be that $r = 4$ and $s = 1$ or vice versa.

The point is that the function f is not quite the same as that which is denoted by the expression "$f(x)$." The latter, which is an abbreviation for "the value of the function f at x," indicates the value or object into which x is mapped by the function f. As we have seen, this omits some information necessary to a complete specification of f. This slight defect of notation is in most usage not very serious, but an indiscriminate use of the notation in considering *operators*, or functions of functions, may lead to some confusion. Suppose, for example, that G is an operator which maps functions into functions and we write

$$Gf(x + 1). \tag{2.1}$$

Two interpretations of this expression can be considered, corresponding to the two possible meanings we might give the product abc in elementary algebra. In this latter case, in the absence of any rule such as scanning from left to right, the expression may mean (1) the result of obtaining $a(b)$ followed by (a(b))c, or it may mean (2) the resulto of obtaining b(c) followed by

$a(b(c))$. The statement that these two results are the same, or that the order does not matter in performing this continued multiplication is, of course, known as the associative law of multiplication. This law holds in multiplication, but there is no reason to think that it is true in all other situations.

In the case of (2.1), we can similarly take it to mean (1) determine the function into which G maps f and then determine the value of this function for the argument $x + 1$, or it may mean (2) take the value of $f(x + 1)$, considered as a function of x, and determine the function into which this is mapped by G.

These two interpretations correspond to writing (1) $(G(f))(x + 1)$ or (2) $G(f(x + 1))$. Unlike ordinary multiplication they may give quite different results. Suppose, for example, that G is the operator that maps the arbitrary function f into the function ϕ, $G(f) = \phi$, where $\phi(x) = xf(x)$. If, then, f is the constant function $f(x) = 1$ (i.e., $f(x)$ is 1 for all values of x), we have $\phi(x) = x$. We thus can write $G(f(x)) = \phi(x)$ or, alternatively, since $\phi(x) = x$ and $f(x) = 1$, write $G(1) = x$.

We then have

$$(G(f))(x + 1) = \phi(x + 1) = x + 1,$$

while on the other hand,

$$G(f(x + 1)) = G(1) = x.$$

A more precise notation, called the lambda [λ] notation, removes the ambiguity present in the usual functional notation by clearly signaling the independent variables in a function definition.

The function $x \to f(x)$ is denoted by $\lambda x f(x)$. Thus $\lambda x(x^2)$ represents the function defined by the equation $f(x) = x^2$; $\lambda x(3x + y)$ specifies that x is the independent argument in the function $g(x) = 3x + y$, and $\lambda x(x^2 + 2x + 1)$ shows clearly that the map $x \to x^2 + 2x + 1$ is intended and not the map $x + 1 \to x^2 + 2x + 1$.

To write an expression like $f(x + 1)$, the result of applying the map f to $x + 1$, where $f(x) = x^2$, we would write

$$(\lambda x(x^2))(x + 1).$$

In so doing, the kind of confusion that was implicit in the standard notation is removed. This last expression denotes the value of the function $f: x \to x^2$ when x is $x + 1$, and not the function $x \to x^2 + 2x + 1$.

A function of several variables $(x_1, x_2, \ldots, x_n) \to f(x_1, x_2, \ldots, x_n)$ is represented by $\lambda x_1 \lambda x_2 \ldots \lambda x_n f(x_1, x_2, \ldots, x_n)$. The sum function for three arguments is then written $\lambda x_1 \lambda x_2 \lambda x_3 (x_1 + x_2 + x_3)$.

A calculus for the manipulation of expressions employing this notation has been developed [2] and is called the "lambda calculus." It is used in one of the several methods of defining the class of computable functions. Some

investigations into the design of computing machines that operate on such lambda expressions have been made [1,5], with proponents advocating significant advantages to such an approach to machine design.

Perhaps a key point of the lambda notation is implicit in the programmed definition of functions. If we consider the definition of a function by a subroutine, the subroutine alone does not constitute the complete definition of the function. The missing ingredient is, of course, the argument or arguments to be used in the computation. Thus in using, say PL/I, we might write

```
CALL FUNCTION(X1, X2);
```

to cause the execution of the subroutine labeled FUNCTION that will use the arguments X1, X2. The subroutine that performs the computation must be given the locations of the arguments that it is to use before it can be executed.

2.9 GENETIC METHODS IN DEFINING SETS

Many of the sets, or collections of objects, that arise within mathematics and the computer sciences have a generative character. That is, in defining a set we may be given certain fundamental objects in the set and, in addition, be given rules for forming new objects from old ones, or from objects already known to be in the set. The defined set then consists of all objects that can be formed by starting with the fundamental objects and repeatedly applying the rules for forming new objects.

The class of all human beings (from the past, present, and future) would probably be, if we accept the theory of evolution, impossible to define sharply. However, as with most beliefs based on orthodoxy, things are considerably simplified if we take Genesis literally. With this hypothesis we can define the class of human beings by writing

```
HUMAN : : = ADAM | EVE | CHILD(HUMAN, HUMAN)
```

In words, understanding that the symbol "|" is to be interpreted as "OR," this notation is to be taken as meaning that a member of the class of human beings is either Adam or Eve or the child of any two human beings. Only Adam and Eve are seen to be humans from a first application. A second application of the rule yields Cain and Abel and their brothers and sisters and then, by repetition, all generations until Armageddon. This particular definition is recursive in that the class of humans is defined in terms of itself with the word HUMAN appearing on both sides of the defining expression.

Such methods of defining collections of objects have been called "ge-

"genetic" by Hilbert [4], and our example shows the appropriatness of the

name. Perhaps the most obvious example of their use in mathematics appears in the definition of the natural numbers. These can be obtained by starting with the number 1 and repeatedly applying a rule that furnishes the successor of a number when we are given that number (see Section 12.2, "Peano's Axioms"). Here, each new member has just one "parent," unlike our biblical example. More generally, the method may presume any number of "parents" in generating new members.

As we shall see, examples of the use of the method abound in the study of programming languages and in computer science in general.

2.10 FORMAL SYSTEMS

During the last 100 years there has been a notable attempt to use mathematical logic to put mathematics on a firmer foundation and thus minimize the risks in the use of intuition by our fallible intellects (cf. Appendix to Chapter 1). A frequent device has been the use of *formal systems*—systems where we manipulate symbolic expressions without regard to their possible meaning. The methods used have either motivated or been rediscovered in the employment of techniques useful for the study of programming languages and their compilers.

By way of example, we briefly describe the first steps in a formal approach to the study of the game of checkers, a game familiar to all readers. We shall do this only in the most superficial manner and shall not at all consider the substantive problem of what constitutes good strategy in playing this game.

Let us first consider the elements, or *objects*, in the game of checkers to be board positions. We designate the two contestants as White and Black where each uses pieces of the corresponding color. Starting with the configuration of Fig. 2.2, Black makes the first move and a game is a sequence of board positions, each of which is obtained from the preceding one as a result of a legitimate move. There are 32 squares on a checkerboard where a piece can be situated (Fig. 2.2). Each such square can contain no piece, a black checker, a black king, a white checker, or a white king. We can then represent each board position P by a string of 33 digits, $P = A_1A_2 \ldots A_{32}B$, where each digit A_i $(i = 1, 2, \ldots, 32)$ corresponds to one of the 32 labeled squares of Fig. 2.2 and can be taken to be, say, 0, 1, 2, 3, or 4 according to the five listed possibilities. B is defined to be 0 or 1 according to whether White or Black is to make the next move. There are a finite number, $2 \cdot 5^{32}$, of such configurations for P, not all of which are *feasible* board positions, that is, positions that can be reached in playing a game. For example, there cannot be 13 black pieces in play. These configurations are the "objects" of our system. The rules of the game imply that if $P = A_1A_2 \ldots A_{32}B$ is a particular board position, then there is a small set of possible board positions that can result from

Fig. 2.2 Checkerboard notation. The numbered and the others are light-colored. Initially, Black occ 21 to 32.

a legitimate move by White or by Black. We $P' = A'_1 A'_2 \ldots A'_{32} B'$ is a successor position to move by writing

$$P \to P'.$$

If a sequence of moves alternately made by Bl game fragment) leads from position P to posi

$$P \to^* Q.$$

It is possible in these terms to play checke

sional array, as just about everyone in the Western world has learned as a child, but rather clumsy to describe in terms of the one-dimensional strings that represent board positions. A "theorem" of the system might be taken to be a statement of the form $P_0 \rightarrow * Q$, indicating that position Q is a feasible board position, one that can be reached in a game.

Having defined this rudimentary prototype of a formal system, we turn to a more detailed description of such a system, appropriate to its use in mathematics. In general, a formal system for the study of some mathematical structure is defined by taking the following steps (1) through (4). We shall illustrate this in a following discussion of Boolean algebra.

1. Identify the basic elements of the structure. For example, in geometry these might be points, lines, and planes; in logic they might be statements (assertions that are either true or false); in the study of the integers they might be the objects zero and one; in programming languages they might be the basic words and symbols that can appear within program statements. Introduce, if necessary, symbols or primitive terms to represent these basic elements and also, as needed, symbols to denote the essential attributes, operations, relations, and so forth that may involve these terms and are of special significance. These symbols can be any identifiers we please—single letters or marks, words, sentences, anything that will serve as a name for that which is being represented. For example, as part of the description of decimal data in a particular programming language, we might (as in the description of ALGOL [7]) describe the formation of the decimal integers by using as primitive terms the symbols that represent the ten digits. We might write

$$<\text{digit}> :: = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$$

Here, the brackets "< >" are taken to mean "the symbolic expressions for the members of the class" named within the brackets. In this case, the class is the class of decimal digits.

The formal notational scheme here illustrated is an example of the "Backus-Naur form," also called the "Backus normal form" (BNF), introduced in the report on ALGOL [7] as a technique for defining programming languages.

2. Describe, as in the discussion of genetic methods, how to form new objects in terms of objects already known to be in the system. Specify this by indicating how to form new strings of symbols, or new terms, that represent these new objects. For example, we might define the representations of all members of the class of nonnegative integers K, where the fundamental objects are the decimal digits, by writing

$$<K> :: = <\text{digit}> \mid <K><\text{digit}>.$$

This means that the symbolic strings representing the nonnegative integers are formed by taking any digit or any symbolic string representing a nonnegative integer and appending a digit to it. These rules admit strings of arbitrary length and with possible leading zeros as representors of the nonnegative integers. We leave to the reader (cf. Exercise 2.15) the changes that would be needed to admit only strings that do not have a leading zero.

As a part of the description of elementary algebra we might let the letters of the alphabet denote variables, which are some of the primitive terms of the system. After introducing the operation symbol + we can specify that if x, y are two terms then $(x + y)$ is a term or, in the BNF notation, we might write

$$<\text{term}> :: = (<\text{term}> + <\text{term}>) \mid <\text{primitive term}>.$$

3. Describe how statements or "formulas" involving the terms and special symbols of the system are formed. These, also, have a genetic or constructive character. Our prescriptions may include recursive rules showing how the symbolic strings representing new formulas can be formed from those representing earlier obtained formulas.

Thus, if the symbols $2, 3, 5, (,), >$ are included among the primitive symbols in a discussion of elementary algebra, $5 > (2 + 3)$ would be a properly formed string of symbols identifying a statement (which happens to be false), but $> 5)23 (+$ would be an ill-formed, meaningless string of these same symbols. The rules describing the formation of well-formed strings comprise the "syntax" of the system. The properly formed strings representing formulas are often called "well-formed formulas" or "wff's." The term is also used to refer to those properly formed strings that represent the objects of the system.

This is analogous to the construction of properly formed statements within a programming language. The syntax of the language specifies how well-formed statements are formed from the basic symbols and words of the language (cf. Chapter 9).

4. Specify certain fundamental statements, or formulas, that are assumed to be true. These are called the "axioms" or "postulates" of the system. Further, specify how from one or more true statements others can be obtained by using principles of logic and special rules that may be peculiar to the system being defined. The true statements so generated are the "theorems" of the system.

A formal system then consists of all these components—a set of admissible symbols, rules for composing wff's, axioms which are the fundamental wff's assumed to be true, and rules for generating new true statements (or theorems) from previously obtained ones.

The Euclidean plane geometry studied in secondary school has some characteristics of such systems, but our dependence on diagrams in the usual exposition of this subject is a serious violation of the specifications for formal systems. Carelessly drawn inferences from diagrams are, in fact, the bases for a number of fallacies of plane geometry that are occasionally seen in works on mathematical recreations.

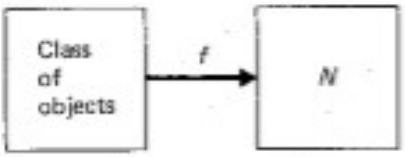
In studying a mathematical structure by the device of a formal system, the system must, in a sense, mirror that structure. The most crucial step is perhaps the enunciation of the axioms. Once the system is laid down, the derivation of the theorems within it can proceed through symbolic manipulation and in game-like fashion with no further reference to the structure it is designed to emulate.

The formal description of a programming language has, as we have indicated, some of the aspects of a formal system. Steps (1), (2), (3) are followed in describing properly formed statements and programs within the language (cf. Chapter 9).

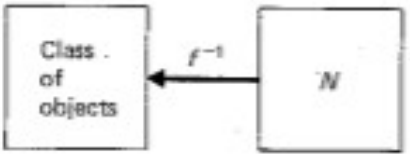
2.11 ARITHMETIZATION

We referred briefly in Chapter 1 to the technique of arithmetization that is used in mathematical logic. We shall elaborate somewhat upon these ideas because of their great significance in computing as well as in logic.

We have noted, in reference to the Pythagoreans, the primacy of number. When, in the very varied applications of the computer, we map many different classes of objects into the natural numbers we are, perhaps, giving further evidence of the primary significance of number in the order of things. In these applications many differing kinds of objects or situations ranging from checks to checkers, from formulas to forests, from payrolls to poems are mapped into the class of natural numbers N (or, perhaps, into sequences of natural numbers).



Since we shall want to be able to "recover" each object from its representation—for example, it would certainly not do to map all of the objects into zero—it is necessary to have our map one-to-one so that no two objects are mapped into the same number. The map is then, as we have noted, invertible.



In the study of a formal system, the technique of arithmetization, when applied, includes the mapping of the terms and formulas of the system into the natural numbers by such invertible functions. Further, (1) the generating operations that yield new terms from existing terms, (2) those operations that obtain new wff's from existing terms and wff's and (3) those operations that yield new theorems from existing theorems can all be replaced by *arithmetic functions* that map the numbers corresponding to the existing terms, wff's, theorems into the numbers corresponding to the new terms, wff's, theorems which are derivable from these existing objects.

This is shown schematically in part in Fig. 2.3, which shows the arithmetization of the derivation of theorems in a formal system.

A similar diagram will show the arithmetization of the derivation of wff's from the terms of the system.

In the arithmetization of the system, on the right-hand side of Fig. 2.3, we work with *numbers* and arithmetic *operations* performed on these numbers. The arithmetization is a *simulation* of the formal system in which the original objects of the system and the operations performed on these objects have been replaced by numbers and arithmetic operations performed on these numbers. The operations that can be performed within the system, shown on the left-hand side of Fig. 2.3, are mirrored by the arithmetic operations that occur on the right-hand side of Fig. 2.3. The arithmetized processes mimic the symbolic manipulations within the system. This is said in the sense that if we perform the mappings in the order $A \to B$ fol-

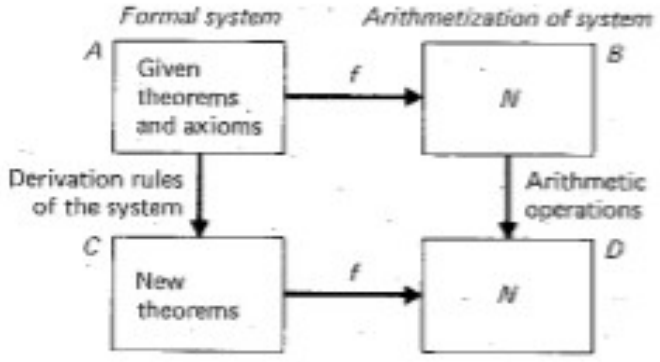


Fig. 2.3 The arithmetization of a formal system.

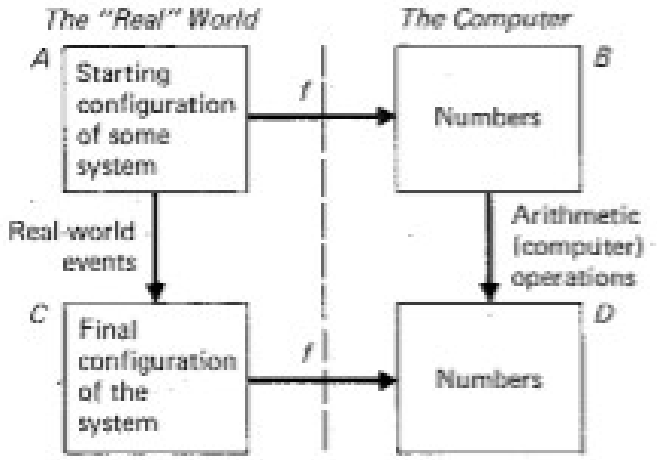


Fig. 2.4 Simulation—mirroring a real world situation in the computer. The simulation is correct if the two paths from A to D lead to the same results.

lowed by $B \rightarrow D$, or in the order $A \rightarrow C$ followed by $C \rightarrow D$ (corresponding to the two paths from A to D), the results are the same. (In mathematics we say that the diagram "commutes.")

The technique is significant in our survey because in a larger sense it is the essence of what is meant by simulation in many applications of computing (see Fig. 2.4). The programmed operations in a simulation process can, although there may be no point to doing it, always be expressed as arithmetic operations. This will follow from the discussion in Chapter 4 where the computer-defined functions are claimed to be the same as a class called the "partial recursive functions."