Part I

Foundations

1

Fundamental concepts and applications

The purpose of this introductory chapter is threefold. First, it contains the main definitions, terminology, and notations that are used throughout the book. After the introduction of our main feature characters – namely, Boolean functions – several sections are devoted to a discussion of alternative representations, or expressions, of Boolean functions. Disjunctive and conjunctive normal forms, in particular, are discussed at length in Sections 1.4–1.11. These special algebraic expressions play a very central role in our investigations, as we frequently focus on the relation between Boolean functions and their normal forms. Section 1.12, however, also provides a short description of different types of function representations, namely, representations over GF(2), pseudo-Boolean polynomial expressions, and binary decision diagrams.

A second objective of this chapter is to introduce several of the topics to be investigated in more depth in subsequent chapters, namely: fundamental algorithmic problems (Boolean equations, generation of prime implicants, dualization, orthogonalization, etc.) and special classes of Boolean functions (bounded-degree normal forms, monotone functions, Horn functions, threshold functions, etc.). Finally, the chapter briefly presents a variety of applications of Boolean functions in such diverse fields as logic, electrical engineering, reliability theory, game theory, combinatorics, and so on. These applications have often provided the primary motivation for the study of the problems to be encountered in the next chapters.

In a sense, this introductory chapter provides a (very) condensed digest of what's to come. It can be considered a degustation: Its main purpose is to whet the appetite, so that readers will decide to embark on the full course!

1.1 Boolean functions: Definitions and examples

This book is about Boolean functions, meaning: $\{0,1\}$-valued functions of a finite number of $\{0,1\}$-valued variables.

Definition 1.1. *A* Boolean function of n variables *is a function on $\mathcal{B}^n$ into $\mathcal{B}$, where $\mathcal{B}$ is the set* $\{0,1\}$*, n is a positive integer, and $\mathcal{B}^n$ denotes the n-fold cartesian product of the set $\mathcal{B}$ with itself. A point $X^* = (x_1, x_2, \ldots, x_n) \in \mathcal{B}^n$ is a* true point (*respectively,* false point) *of the Boolean function f if $f(X^*) = 1$ (respectively, $f(X^*) = 0$). We denote by $T(f)$ (respectively, $F(f)$) the set of* true points (*respectively,* false points) *of f. We denote by $\mathbb{1}_n$ the function that takes constant value* 1 *on $\mathcal{B}^n$ and by $\mathbb{0}_n$ the function that takes constant value* 0 *on $\mathcal{B}^n$.*

It should be stressed that, in many applications, the role of the set $\mathcal{B}$ is played by another two-element set, like {Yes, No}, {True, False}, {ON, OFF}, {Success, Failure}, $\{-1, 1\}$ or, more generally, $\{a, b\}$, where a and b are abstract (uninterpreted) elements. In most cases, this distinction is completely irrelevant. However, it is often convenient to view the elements of $\mathcal{B}$ as *numerical quantities* in order to perform arithmetic operations on these elements and to manipulate algebraic expressions like $1 - x$, $x + y - xy$, and so on, where x, y are elements of $\mathcal{B}$.

As an historical aside, it is interesting to note that the ability to perform algebraic computations on logical symbols, in a way that is at least formally similar to what we are used to doing for numerical quantities, was one of the driving forces behind George Boole's seminal work in logic theory. Let us quote from Boole [103], Chapter V.6 (italics are Boole's):

> [...] any system of propositions may be expressed by equations involving symbols x, y, z, which, whenever interpretation is possible, are subject to laws identical in form with the laws of a system of quantitative symbols, susceptible only of the values 0 and 1. But as the formal processes of reasoning depend only upon the laws of the symbols, and not upon the nature of their interpretation, we are permitted to treat the above symbols, x, y, z, as if they were quantitative symbols of the kind above described. *We may in fact lay aside the logical interpretation of the symbols in the given equation; convert them into quantitative symbols, susceptible only of the values* 0 *and* 1*; perform upon them as such all the requisite processes of solution; and finally restore to them their logical interpretation.* And this is the mode of procedure which will actually be adopted [...]

In this book, we systematically follow Boole's prescription and adhere to the convention that $\mathcal{B} = \{0, 1\}$, where 0 and 1 can be viewed as either abstract symbols or numerical quantities.

The most elementary way to define a Boolean function f is to provide its *truth table*.

Definition 1.2. *The* truth table *of a Boolean function on $\mathcal{B}^n$ is a complete list of all the points in $\mathcal{B}^n$ together with the value of the function at each point.*

Example 1.1. *The truth table of a Boolean function on $\mathcal{B}^3$ is shown in Table 1.1.* □

Of course, the use of truth tables becomes extremely cumbersome when the function to be defined depends on more than, say, 5 or 6 arguments. As a matter

Table 1.1. Truth Table for Example 1.1

(x_1,x_2,x_3)	$f(x_1,x_2,x_3)$
(0,0,0)	1
(0,0,1)	1
(0,1,0)	0
(0,1,1)	1
(1,0,0)	0
(1,0,1)	1
(1,1,0)	0
(1,1,1)	1

of fact, Boolean functions are often defined implicitly rather than explicitly, in the sense that they are described through a procedure that allows us, for any $0-1$ point in the domain of interest, to compute the value of the function at this point. In some theoretical developments, or when we analyze the computational complexity of certain problems, such a procedure can simply be viewed as a *black box oracle*, of which we can observe the output (that is, the function value) for any given input, but not the inner working (that is, the details of the algorithm that computes the output). In most applications, however, more information is available regarding the process that generates the function of interest, as illustrated by the examples below. (We come back to these applications in much greater detail in Section 1.13 and in many subsequent chapters of the book.)

Application 1.1. (Logic.) *In many applications (such as those arising in artificial intelligence), a Boolean function can be viewed as indicating the* truth value *of a sentence of propositional (or Boolean) logic. Consider, for instance, the sentence* S*: "If it rains in the morning, or if the sky is cloudy, then I carry my umbrella." Let us denote by x_1, x_2, and x_3, respectively, the subsentences "it rains in the morning," "the sky is cloudy," and "I carry my umbrella". Then,* S *can be identified with the sentence*

$$(x_1 \text{ } OR \text{ } x_2) \Rightarrow x_3.$$

It is easy to see that the function displayed in Table 1.1 computes the truth value of S for all possible values of x_1, x_2, x_3, under the usual correspondence True $\leftrightarrow$ 1, False $\leftrightarrow$ 0. □

Application 1.2. (Electrical engineering.) *In electrical or in computer engineering, a switching circuit is often abstracted into the following model, called a* combinational circuit. *The wiring of the circuit is described by an acyclic directed graph $D = (V,A)$. The vertices of D are the* gates *of the circuit. The indegree of each gate is at most 2. Each gate with indegree 2 is labeled either AND or OR, and each gate with indegree 1 is labeled NOT. The gates with indegree 0 are called* input gates *and are denoted $v_1, v_2, \dots, v_n$. Also, all gates of D have outdegree 1, except for a single gate f, called* output gate*, which has outdegree 0.*

Every such circuit can be viewed as representing a Boolean function $f_D(x_1,x_2,\dots,x_n)$*. First, for every* $(x_1,x_2,\dots,x_n) \in \mathcal{B}^n$*, the* state $s(v)$ *of gate* $v \in V$ *is computed according to the following recursive rules:*

1. *For each input gate* v_i, $s(v_i) = x_i$ $(i = 1,2,\dots,n)$.
2. *For each AND-gate* $v \in V$*, if* $(u,v),(w,v) \in A$ *are the arcs entering* v*, then* $s(v) = \min(s(u),s(w))$.
3. *For each OR-gate* $v \in V$*, if* $(u,v),(w,v) \in A$ *are the arcs entering* v*, then* $s(v) = \max(s(u),s(w))$.
4. *For each NOT-gate* $v \in V$*, if* $(u,v) \in A$ *is the arc entering* v*, then* $s(v) = 1 - s(u)$*. Finally, we let* $f_D(x_1,x_2,\dots,x_n) = s(f)$.

For instance, the circuit represented in Figure 1.1 computes the function given in Example 1.1. This can easily be verified by computing the state of the output gate (in this case, the OR-gate) for all possible 0–1 *inputs. For example, if* $(x_1,x_2,x_3) = (0,0,0)$*, then one successively finds that the state of each NOT-gate is* 1 *(*$= 1-0$*); the state of the AND-gate is* 1 *(*$= \min(1,1)$*); and the state of the output gate is* 1 *(*$= \max(1,0)$*).*

More generally, the gates of a combinational circuit may be "primitive" Boolean functions forming another class from the {AND,OR,NOT} collection used in our small example. In all cases, the gates may be viewed as atomic units of hardware, providing the building blocks for the construction of larger circuits. □

Historically, propositional logic and electrical engineering have been the main nurturing fields for the development of research on Boolean functions. However, because they are such fundamental mathematical objects, Boolean functions have also been used to model a large number of applications in a variety of areas. To describe these applications, we introduce a few more notations.

Given a point $X \in \mathcal{B}^n$, we denote by $supp(X)$ the *support* of X, that is, $supp(X)$ is the set $\{i \in \{1,2,\dots,n\} \mid x_i = 1\}$. (Conversely, X is the *characteristic vector* of $supp(X)$.)

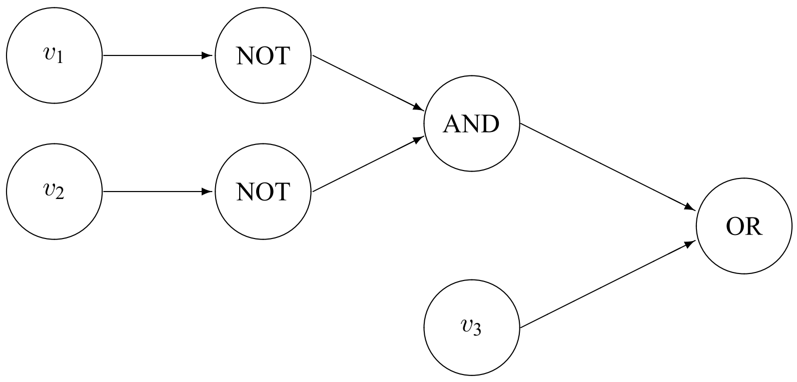

Figure 1.1. A small combinational circuit.

Application 1.3. (Game theory.) *Many group decision procedures (such as those used in legislative assemblies or in corporate stockholder meetings) can be viewed, in abstract terms, as decision rules that associate a single dichotomous "Yes–No" outcome (for instance, adoption or rejection of a resolution) with a collection of dichotomous "Yes–No" votes (for instance, assent or disagreement of individual lawmakers). Such procedures have been studied in the game-theoretic literature under the name of* simple games *or* voting games. *More formally, let* $N = \{1,2,\ldots,n\}$ *be a finite set, the elements of which are to be called* players. *A simple game on* N *is a function* $v : \{A \mid A \subseteq N\} \to \mathcal{B}$. *Clearly, from our vantage point, a simple game can be equivalently modeled as a Boolean function* f_v *on* $\mathcal{B}^n$*: The variables of* f_v *are in 1-to-1 correspondence with the players of the game (variable i takes value* 1 *exactly when player i votes "Yes"), and the value of the function reflects the outcome of the vote for each point* $X^* \in \mathcal{B}^n$ *describing a vector of individual votes:*

$$f_v(X^*) = \begin{cases} 1 & \textit{if } v(supp(X^*)) = 1, \\ 0 & \textit{otherwise.} \end{cases}$$

□

Application 1.4. (Reliability theory.) *Reliability theory investigates the relationship between the operating state of a complex system S and the operating state of its individual components, say components* $1,2,\ldots,n$. *It is commonly assumed that the system and its components can be in either of two states:* operative *or* failed. *Moreover, the state of the system is completely determined by the state of its components via a deterministic rule embodied in a Boolean function* f_S *on* $\mathcal{B}^n$*, called the* structure function *of the system: For each* $X^* \in \mathcal{B}^n$,

$$f_S(X^*) = \begin{cases} 1 & \textit{if the system operates when all components in } supp(X^*) \textit{ operate} \\ & \textit{and all other components fail,} \\ 0 & \textit{otherwise.} \end{cases}$$

A central issue is to compute the probability that the system operates (meaning that f_S *takes value* 1*) when each component is subject to probabilistic failure. Thus, reliability theory deals primarily with the stochastic theory of Boolean functions.* □

Application 1.5. (Combinatorics.) *Consider a* hypergraph $\mathcal{H} = (N,\mathcal{E})$*, where* $N = \{1,2,\ldots,n\}$ *is the set of* vertices *of* $\mathcal{H}$*, and* $\mathcal{E}$ *is a collection of subsets of* N*, called* edges *of the hypergraph. A subset of vertices is said to be* stable *if it does not contain any edge of* $\mathcal{H}$*. With* $\mathcal{H}$*, we associate the Boolean function* $f_{\mathcal{H}}$ *defined as follows: For each* $X^* \in \mathcal{B}^n$,

$$f_{\mathcal{H}}(X^*) = \begin{cases} 1 & \textit{if } supp(X^*) \textit{ is not stable,} \\ 0 & \textit{otherwise.} \end{cases}$$

The function $f_{\mathcal{H}}$ *is the* stability function *of* $\mathcal{H}$. □

Of course, the kinship among the models presented in Applications 1.3–1.5 is striking: It is immediately apparent that we are really dealing here with a single class of mathematical objects, in spite of the distinct motivations that originally justified their investigation.

Applications of Boolean functions will be discussed more thoroughly in Section 1.13, after we have introduced some of the fundamental theoretical concepts that underlie them.

Before we close this section, let us add that, in this book, our view of Boolean functions will be mostly combinatorial and algorithmic. For algebraic or logic-oriented treatments, we refer the reader to the excellent books by Rudeanu [795, 796] or Brown [156]. In these books, as in many related classical publications by other authors, Boolean functions are actually defined more broadly than in Definition 1.1, as (special) mappings of the form $f : \mathcal{A}^n \to \mathcal{A}$, where $\mathcal{A}$ is the carrier of an *arbitrary* Boolean algebra $(\mathcal{A}, \cup, \cap, \neg, 0, 1)$. By contrast, we shall essentially restrict ourselves in this book to the two-element Boolean algebra $(\mathcal{B}, \vee, \wedge, \overline{\cdot}, 0, 1)$, where $\mathcal{B} = \{0, 1\}$ (see Section 1.2). Brown [156], in particular, discusses in great detail the pros and cons of working with two-element, rather than more general, Boolean algebras. While acknowledging the relevance of his arguments, we feel that, at the risk of giving up some generality, our restricted framework is already sufficiently rich to model a variety of interesting applications and to allow us to handle a host of challenging algorithmic problems of a combinatorial nature. Also, the terminology introduced in Definition 1.1 has become sufficiently entrenched to justify its continued use, rather than the alternative terminology *switching functions* or *truth functions* which, though less liable to create confusion, has progressively become obsolete.

1.2 Boolean expressions

As the above examples illustrate, Boolean functions can be described in many alternative ways. In this section, we concentrate on a type of representation derived from propositional logic, namely, the representation of Boolean functions by *Boolean expressions* (see, for instance, [156, 680, 795, 848] for different presentations).

Boolean expressions will be used extensively throughout the book. In fact, the emphasis on Boolean expressions (rather than truth tables, circuits, oracles, etc.) can be seen as a main distinguishing feature of our approach and will motivate many of the issues we will tackle in subsequent chapters.

Our definition of Boolean expressions will be recursive, starting with three elementary operations as building blocks.

Definition 1.3. *The binary operation* $\vee$ (disjunction, Boolean OR), *the binary operation* $\wedge$ (conjunction, Boolean AND), *and the unary operation* $\overline{\cdot}$ (complementation, negation, Boolean NOT) *are defined on* $\mathcal{B}$ *by the following rules:*

$$0 \vee 0 = 0, \;\; 0 \vee 1 = 1, \;\; 1 \vee 0 = 1, \;\; 1 \vee 1 = 1;$$

$$0 \wedge 0 = 0, \;\; 0 \wedge 1 = 0, \;\; 1 \wedge 0 = 0, \;\; 1 \wedge 1 = 1;$$

$$\overline{0} = 1, \;\; \overline{1} = 0.$$

For a Boolean variable x, we sometimes use the following convenient notation:

$$x^\alpha = \begin{cases} x, & \text{if } \alpha = 1, \\ \overline{x}, & \text{if } \alpha = 0. \end{cases}$$

Keeping in line with our focus on functions, we often regard the three elementary Boolean operations as defining Boolean functions on $\mathcal{B}^2$: $\text{disj}(x,y) = x \vee y$, $\text{conj}(x,y) = x \wedge y$, and on $\mathcal{B}$: $\text{neg}(x) = \overline{x}$. When the elements of $\mathcal{B} = \{0,1\}$ are interpreted as integers rather than abstract symbols, these operations can be defined by simple arithmetic expressions: For all $x, y \in \mathcal{B}$,

$$x \vee y = \max\{x,y\} = x + y - x\,y,$$

$$x \wedge y = \min\{x,y\} = x\,y,$$

$$\overline{x} = 1 - x.$$

Observe that the conjunction of two elements of $\mathcal{B}$ is equal to their arithmetic product. By analogy with the usual convention for products, we often omit the operator $\wedge$ and denote conjunction by mere juxtaposition.

We can extend the definitions of all three elementary operators to $\mathcal{B}^n$ by writing: For all $X, Y \in \mathcal{B}^n$,

$$X \vee Y = (x_1 \vee y_1, x_2 \vee y_2, \ldots, x_n \vee y_n),$$

$$X \wedge Y = (x_1 \wedge y_1, x_2 \wedge y_2, \ldots, x_n \wedge y_n) = (x_1 y_1, x_2 y_2, \ldots, x_n y_n),$$

$$\overline{X} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n).$$

Let us enumerate some of the elementary properties of disjunction, conjunction, and complementation. (We note for completeness that the properties listed in Theorem 1.1 can be viewed as the defining properties of a general Boolean algebra.)

Theorem 1.1. *For all $x, y, z \in \mathcal{B}$, the following identities hold:*

(1) $x \vee 1 = 1$ *and* $x \wedge 0 = 0$;
(2) $x \vee 0 = x$ *and* $x \wedge 1 = x$;
(3) $x \vee y = y \vee x$ *and* $x\,y = y\,x$ (commutativity);
(4) $(x \vee y) \vee z = x \vee (y \vee z)$ *and* $x\,(y\,z) = (x\,y)\,z$ (associativity);
(5) $x \vee x = x$ *and* $x\,x = x$ (idempotency);
(6) $x \vee (x\,y) = x$ *and* $x\,(x \vee y) = x$ (absorption);
(7) $x \vee (y\,z) = (x \vee y)\,(x \vee z)$ *and* $x\,(y \vee z) = (x\,y) \vee (x\,z)$ (distributivity);
(8) $x \vee \overline{x} = 1$ *and* $x\,\overline{x} = 0$;
(9) $\overline{\overline{x}} = x$ (involution);
(10) $\overline{(x \vee y)} = \overline{x}\,\overline{y}$ *and* $\overline{(x\,y)} = \overline{x} \vee \overline{y}$ (De Morgan's laws);
(11) $x \vee (\overline{x}\,y) = x \vee y$ *and* $x\,(\overline{x} \vee y) = x\,y$ (Boolean absorption).

Proof. These identities are easily verified, for example, by exhausting all possible values for x, y, z. □

Building upon Definition 1.3, we are now in a position to introduce the important notion of *Boolean expression.*

Definition 1.4. *Given a finite collection of* Boolean variables $x_1, x_2, \dots, x_n$, *a* Boolean expression (*or* Boolean formula) in the variables $x_1, x_2, \dots, x_n$ *is defined as follows:*

(1) *The constants* $0, 1$, *and the variables* $x_1, x_2, \dots, x_n$ *are Boolean expressions in* $x_1, x_2, \dots, x_n$.
(2) *If* ϕ *and* ψ *are Boolean expressions in* $x_1, x_2, \dots, x_n$, *then* $(\phi \vee \psi)$, $(\phi\psi)$ *and* $\overline{\phi}$ *are Boolean expressions in* $x_1, x_2, \dots, x_n$.
(3) *Every Boolean expression is formed by finitely many applications of the rules* (1)–(2).

We also say that a Boolean expression in the variables $x_1, x_2, \dots, x_n$ *is a* Boolean expression on $\mathcal{B}^n$.

We use notations like $\phi(x_1, x_2, \dots, x_n)$ or $\psi(x_1, x_2, \dots, x_n)$ to denote Boolean expressions in the variables $x_1, x_2, \dots, x_n$.

Example 1.2. *Here are some examples of Boolean expressions:*

$\phi_1(x) = x$,
$\phi_2(x) = \overline{x}$,
$\psi_1(x, y, z) = (((\overline{x} \vee y)(y \vee \overline{z})) \vee ((xy)z))$,
$\psi_2(x_1, x_2, x_3, x_4) = ((x_1 x_2) \vee (\overline{x}_3 \overline{x}_4))$. □

Now, since disjunction, conjunction, and complementation can be interpreted as Boolean functions, every Boolean expression $\phi(x_1, x_2, \dots, x_n)$ can also be viewed as generating a Boolean function defined by composition.

Definition 1.5. *The Boolean function* f_ϕ represented (*or* expressed) *by a Boolean expression* $\phi(x_1, x_2, \dots, x_n)$ *is the unique Boolean function on* $\mathcal{B}^n$ *defined as follows: For every point* $(x_1^*, x_2^*, \dots, x_n^*) \in \mathcal{B}^n$, *the value of* $f_\phi(x_1^*, x_2^*, \dots, x_n^*)$ *is obtained by substituting* x_i^* *for* x_i $(i = 1, 2, \dots, n)$ *in the expression* ϕ *and by recursively applying Definition 1.3 to compute the value of the resulting expression.*

When $f = f_\phi$ *on* $\mathcal{B}^n$, *we also say that* f admits the representation *or the* expression ϕ, *and we simply write* $f = \phi$.

Example 1.3. *Consider again the expressions defined in Example 1.2. We can compute, for instance:*

$f_{\phi_1}(0) = 0, \; f_{\phi_1}(1) = 1$,
$f_{\phi_2}(0) = \overline{0} = 1, \; f_{\phi_2}(1) = \overline{1} = 0$,
$f_{\psi_1}(0,0,0) = (((\overline{0} \vee 0)(0 \vee \overline{0})) \vee ((00)0)) = 1, \dots$

In fact, the expression ψ_1 *in Example 1.2 represents the function* f*, where*

$$f(0,0,1) = f(1,0,0) = f(1,0,1) = 0,$$

$$f(0,0,0) = f(0,1,0) = f(0,1,1) = f(1,1,0) = f(1,1,1) = 1.$$

Thus, we can write

$$f(x,y,z) = \psi_1(x,y,z) = (((\overline{x} \vee y)(y \vee \overline{z})) \vee ((xy)z)). \qquad \square$$

Remark. So that we can get rid of parentheses when writing Boolean expressions, we assume from now on a priority ranking of the elementary operations: Namely, we assume that disjunction has lower priority than conjunction, which has lower priority than complementation. When we compute the value of a parentheses-free expression, we always start with the operations of highest priority: First, all complementations; next, all conjunctions; and finally, all disjunctions. (This is similar to the convention that assigns a lower priority to addition than to multiplication, and to multiplication than to exponentiation when evaluating an arithmetic expression like $3x^2 + 5xy$.) Moreover, we also discard any parentheses that become redundant as a consequence of the associativity property of disjunction and conjunction (Theorem 1.1).

Example 1.4. *The expression* ψ_1 *in Example 1.2 (and hence, the function* f *in Example 1.3) can be rewritten with fewer parentheses as* $f(x,y,z) = \psi_1(x,y,z) = (\overline{x} \vee y)(y \vee \overline{z}) \vee xyz$*. Similarly, the expression* ψ_2 *in Example 1.2 can be rewritten as* $\psi_2(x_1,x_2,x_3,x_4) = x_1x_2 \vee \overline{x}_3\overline{x}_4$. $\square$

The relation between Boolean expressions and Boolean functions, as spelled out in Definition 1.5, deserves to be carefully pondered.

On one hand, it is important to understand that every Boolean function can be represented by *numerous* Boolean expressions (see, for instance, Theorem 1.4 in the next section). In fact, it is easy to see that there are "only" 2^{2^n} Boolean functions of n variables, while there are infinitely many Boolean expressions in n variables. These remarks motivate the distinction we draw between functions and expressions.

On the other hand, since every Boolean expression ϕ represents a *unique* Boolean function f_ϕ, we are justified in interpreting ϕ itself as a function, and we frequently do so. The notation $f = \phi$ introduced in Definition 1.5, in particular, may initially seem confusing, since it equates a function with a formal expression, but this notational convention is actually innocuous: It is akin to the convention for real-valued functions of real variables, where it is usual to assimilate a function with its analytical expression and to write, for instance, equalities like

$$f(x,y) = x^2 + 2xy + y^2 = (x+y)^2. \qquad (1.1)$$

As a matter of fact, since Definition 1.5 implies that we write both $f = \psi$ and $f = \phi$ when ψ and ϕ represent the same function f (compare with Equation (1.1)), it also naturally leads to the next notion.