



Boolean algebra

For other uses, see [Boolean algebra \(disambiguation\)](#).

In [mathematics](#) and [mathematical logic](#), **Boolean algebra** is the branch of [algebra](#) in which the values of the [variables](#) are the [truth values](#) *true* and *false*, usually denoted 1 and 0 respectively. Instead of [elementary algebra](#) where the values of the variables are numbers, and the prime operations are addition and multiplication, the main operations of Boolean algebra are the [conjunction](#) *and* denoted as \wedge , the [disjunction](#) *or* denoted as \vee , and the [negation](#) *not* denoted as \neg . It is thus a formalism for describing logical relations in the same way that elementary algebra describes numeric relations.

Boolean algebra was introduced by [George Boole](#) in his first book *The Mathematical Analysis of Logic* (1847), and set forth more fully in his *An Investigation of the Laws of Thought* (1854).^[1] According to [Huntington](#), the term "Boolean algebra" was first suggested by [Sheffer](#) in 1913,^[2] although [Charles Sanders Peirce](#) in 1880 gave the title "A Boolean Algebra with One Constant" to the first chapter of his "The Simplest Mathematics".^[3] Boolean algebra has been fundamental in the development of [digital electronics](#), and is provided for in all modern programming languages. It is also used in [set theory](#) and [statistics](#).^[4]

History

Boole's algebra predated the modern developments in [abstract algebra](#) and [mathematical logic](#); it is however seen as connected to the origins of both fields.^[5] In an abstract setting, Boolean algebra was perfected in the late 19th century by [Jevons](#), [Schröder](#), [Huntington](#), and others until it reached the modern conception of an (abstract) [mathematical structure](#).^[5] For example, the empirical observation that one can manipulate expressions in the [algebra of sets](#) by translating them into expressions in Boole's algebra is explained in modern terms by saying that the algebra of sets is *a* [Boolean algebra](#) (note the [indefinite article](#)). In fact, [M. H. Stone](#) proved in 1936 that every Boolean algebra is [isomorphic](#) to a [field of sets](#).

In the 1930s, while studying [switching circuits](#), [Claude Shannon](#) observed that one could also apply the rules of Boole's algebra in this setting, and he introduced **switching algebra** as a way to analyze and design circuits by algebraic means in terms of [logic gates](#). Shannon already had at his disposal the abstract mathematical apparatus, thus he cast his switching algebra as the [two-element Boolean algebra](#). In circuit engineering settings today, there is little need to consider other Boolean algebras, thus "switching algebra" and "Boolean algebra" are often used interchangeably.^{[6][7][8]} [Efficient implementation](#) of [Boolean functions](#) is a fundamental problem in the [design](#) of [combinational logic](#) circuits. Modern [electronic design automation](#) tools for [VLSI circuits](#) often rely on an efficient representation of Boolean functions known as (reduced ordered) [binary decision diagrams](#) (BDD) for [logic synthesis](#) and [formal verification](#).^[9]

Logic sentences that can be expressed in [classical propositional calculus](#) have an [equivalent expression](#) in Boolean algebra. Thus, **Boolean logic** is sometimes used to denote propositional calculus performed in this way.^{[10][11][12]} Boolean algebra is not sufficient to capture logic formulas using [quantifiers](#), like those from [first order logic](#). Although the development of mathematical logic did not follow Boole's program, the connection between his algebra and logic was later put on firm ground in the setting of [algebraic logic](#), which also studies the algebraic systems of many other logics.^[5] The problem of [determining whether](#) the variables of a given Boolean (propositional) formula can be assigned in such a way as to make the formula evaluate to true is called the [Boolean satisfiability problem](#) (SAT), and is of importance to [theoretical](#)



Values

Whereas in elementary algebra expressions denote mainly numbers, in Boolean algebra they denote the truth values *false* and *true*. These values are represented with the bits (or binary digits), namely 0 and 1. They do not behave like the integers 0 and 1, for which $1 + 1 = 2$, but may be identified with the elements of the two-element field $\text{GF}(2)$, that is, integer arithmetic modulo 2, for which $1 + 1 = 0$. Addition and multiplication then play the Boolean roles of XOR (exclusive-or) and AND (conjunction) respectively, with disjunction $x \vee y$ (inclusive-or) definable as $x + y - xy$.

Boolean algebra also deals with functions which have their values in the set $\{0, 1\}$. A sequence of bits is a commonly used such function. Another common example is the subsets of a set E : to a subset F of E is associated the indicator function that takes the value 1 on F and 0 outside F . The most general example is the elements of a Boolean algebra, with all of the foregoing being instances thereof.

As with elementary algebra, the purely equational part of the theory may be developed without considering explicit values for the variables.^[13]

Operations

Basic operations

The basic operations of Boolean algebra are as follows:

- AND (conjunction), denoted $x \wedge y$ (sometimes x AND y or Kxy), satisfies $x \wedge y = 1$ if $x = y = 1$ and $x \wedge y = 0$ otherwise.
- OR (disjunction), denoted $x \vee y$ (sometimes x OR y or Axy), satisfies $x \vee y = 0$ if $x = y = 0$ and $x \vee y = 1$ otherwise.
- NOT (negation), denoted $\neg x$ (sometimes NOT x , Nx or $!x$), satisfies $\neg x = 0$ if $x = 1$ and $\neg x = 1$ if $x = 0$.

Alternatively the values of $x \wedge y$, $x \vee y$, and $\neg x$ can be expressed by tabulating their values with truth tables as follows.

x	y	$x \wedge y$	$x \vee y$	x	$\neg x$
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1		
1	1	1	1		

If the truth values 0 and 1 are interpreted as integers, these operations may be expressed with the ordinary operations of arithmetic (where $x + y$ uses addition and xy uses multiplication), or by the minimum/maximum functions:

$$x \wedge y = xy = \min(x, y)$$

$$x \vee y = x + y - xy = \max(x, y)$$

$$\neg x = 1 - x$$



$$x \wedge y = \neg(\neg x \vee \neg y)$$

$$x \vee y = \neg(\neg x \wedge \neg y)$$

Secondary operations

The three Boolean operations described above are referred to as basic, meaning that they can be taken as a basis for other Boolean operations that can be built up from them by **composition**, the manner in which operations are combined or compounded. Operations composed from the basic operations include the following examples:

$$x \rightarrow y = \neg x \vee y$$

$$x \oplus y = (x \vee y) \wedge \neg(x \wedge y)$$

$$x \equiv y = \neg(x \oplus y)$$

These definitions give rise to the following truth tables giving the values of these operations for all four possible inputs.

x	y	$x \rightarrow y$	$x \oplus y$	$x \equiv y$
0	0	1	0	1
1	0	0	1	0
0	1	1	1	0
1	1	1	0	1

The first operation, $x \rightarrow y$, or Cxy , is called **material implication**. If x is true then the value of $x \rightarrow y$ is taken to be that of y . But if x is false then the value of y can be ignored; however the operation must return *some* boolean value and there are only two choices. So by definition, $x \rightarrow y$ is *true* when x is false. (Relevance logic suggests this definition by viewing an implication with a false premise as something other than either true or false.)

The second operation, $x \oplus y$, or Jxy , is called **exclusive or** (often abbreviated as XOR) to distinguish it from disjunction as the inclusive kind. It excludes the possibility of both x and y . Defined in terms of arithmetic it is addition mod 2 where $1 + 1 = 0$.

The third operation, the complement of exclusive or, is **equivalence** or Boolean equality: $x \equiv y$, or Exy , is true just when x and y have the same value. Hence $x \oplus y$ as its complement can be understood as $x \neq y$, being true just when x and y are different. Equivalence's counterpart in arithmetic mod 2 is $x + y + 1$.

Given two operands, each with two possible values, there are $2^2 = 4$ possible combinations of inputs. Because each output can have two possible values, there are a total of $2^4 = 16$ possible binary Boolean operations.

Laws

A **law** of Boolean algebra is an identity such as $x \vee (y \vee z) = (x \vee y) \vee z$ between two Boolean terms, where a **Boolean term** is defined as an expression built up from variables and the constants 0 and 1 using the operations \wedge , \vee , and \neg . The concept can be extended to terms involving other Boolean operations such as \oplus , \rightarrow , and \equiv , but such extensions are



Monotone laws

Boolean algebra satisfies many of the same laws as ordinary algebra when one matches up \vee with addition and \wedge with multiplication. In particular the following laws are common to both kinds of algebra:^[14]

Associativity of \vee :	$x \vee (y \vee z) = (x \vee y) \vee z$
Associativity of \wedge :	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Commutativity of \vee :	$x \vee y = y \vee x$
Commutativity of \wedge :	$x \wedge y = y \wedge x$
Distributivity of \wedge over \vee :	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
Identity for \vee :	$x \vee 0 = x$
Identity for \wedge :	$x \wedge 1 = x$
Annihilator for \wedge :	$x \wedge 0 = 0$

The following laws hold in Boolean Algebra, but not in ordinary algebra:

Annihilator for \vee :	$x \vee 1 = 1$
Idempotence of \vee :	$x \vee x = x$
Idempotence of \wedge :	$x \wedge x = x$
Absorption 1:	$x \wedge (x \vee y) = x$
Absorption 2:	$x \vee (x \wedge y) = x$
Distributivity of \vee over \wedge :	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Taking $x = 2$ in the third law above shows that it is not an ordinary algebra law, since $2 \times 2 = 4$. The remaining five laws can be falsified in ordinary algebra by taking all variables to be 1, for example in Absorption Law 1 the left hand side would be $1(1+1) = 2$ while the right hand side would be 1, and so on.

All of the laws treated so far have been for conjunction and disjunction. These operations have the property that changing either argument either leaves the output unchanged or the output changes in the same way as the input. Equivalently, changing any variable from 0 to 1 never results in the output changing from 1 to 0. Operations with this property are said to be **monotone**. Thus the axioms so far have all been for monotonic Boolean logic. Nonmonotonicity enters via complement \neg as follows.^[4]

Nonmonotone laws

The complement operation is defined by the following two laws.

Complementation 1	$x \wedge \neg x = 0$
Complementation 2	$x \vee \neg x = 1$



the double negation law (also called involution law)

$$\text{Double negation} \quad \neg(\neg x) = x$$

But whereas *ordinary algebra* satisfies the two laws

$$\begin{aligned} (-x)(-y) &= xy \\ (-x) + (-y) &= -(x + y) \end{aligned}$$

Boolean algebra satisfies De Morgan's laws:

$$\text{De Morgan 1} \quad \neg x \wedge \neg y = \neg(x \vee y)$$

$$\text{De Morgan 2} \quad \neg x \vee \neg y = \neg(x \wedge y)$$

Completeness

The laws listed above define Boolean algebra, in the sense that they entail the rest of the subject. The laws *Complementation* 1 and 2, together with the monotone laws, suffice for this purpose and can therefore be taken as one possible *complete* set of laws or axiomatization of Boolean algebra. Every law of Boolean algebra follows logically from these axioms. Furthermore, Boolean algebras can then be defined as the models of these axioms as treated in the section thereon.

To clarify, writing down further laws of Boolean algebra cannot give rise to any new consequences of these axioms, nor can it rule out any model of them. In contrast, in a list of some but not all of the same laws, there could have been Boolean laws that did not follow from those on the list, and moreover there would have been models of the listed laws that were not Boolean algebras.

This axiomatization is by no means the only one, or even necessarily the most natural given that we did not pay attention to whether some of the axioms followed from others but simply chose to stop when we noticed we had enough laws, treated further in the section on axiomatizations. Or the intermediate notion of axiom can be sidestepped altogether by defining a Boolean law directly as any **tautology**, understood as an equation that holds for all values of its variables over 0 and 1. All these definitions of Boolean algebra can be shown to be equivalent.

Duality principle

Principle: If $\{X, R\}$ is a poset, then $\{X, R(\text{inverse})\}$ is also a poset.

There is nothing magical about the choice of symbols for the values of Boolean algebra. We could rename 0 and 1 to say α and β , and as long as we did so consistently throughout it would still be Boolean algebra, albeit with some obvious cosmetic differences.

But suppose we rename 0 and 1 to 1 and 0 respectively. Then it would still be Boolean algebra, and moreover operating on the same values. However it would not be identical to our original Boolean algebra because now we find \vee behaving the way \wedge used to do and vice versa. So there are still some cosmetic differences to show that we've been fiddling with the

now there is no trace of what we have done. The end product is completely indistinguishable from what we started with. We might notice that the columns for $x\wedge y$ and $x\vee y$ in the truth tables had changed places, but that switch is immaterial.

When values and operations can be paired up in a way that leaves everything important unchanged when all pairs are switched simultaneously, we call the members of each pair **dual** to each other. Thus 0 and 1 are dual, and \wedge and \vee are dual. The **Duality Principle**, also called De Morgan duality, asserts that Boolean algebra is unchanged when all dual pairs are interchanged.

One change we did not need to make as part of this interchange was to complement. We say that complement is a **self-dual** operation. The identity or do-nothing operation x (copy the input to the output) is also self-dual. A more complicated example of a self-dual operation is $(x\wedge y) \vee (y\wedge z) \vee (z\wedge x)$. There is no self-dual binary operation that depends on both its arguments. A composition of self-dual operations is a self-dual operation. For example, if $f(x, y, z) = (x\wedge y) \vee (y\wedge z) \vee (z\wedge x)$, then $f(f(x, y, z), x, t)$ is a self-dual operation of four arguments x,y,z,t .

The principle of duality can be explained from a group theory perspective by the fact that there are exactly four functions that are one-to-one mappings (automorphisms) of the set of Boolean polynomials back to itself: the identity function, the complement function, the dual function and the contradual function (complemented dual). These four functions form a group under function composition, isomorphic to the Klein four-group, acting on the set of Boolean polynomials. Walter Gottschalk remarked that consequently a more appropriate name for the phenomenon would be the *principle* (or *square*) *of quaternality*.^[15]

Diagrammatic representations

Venn diagrams

A Venn diagram^[16] is a representation of a Boolean operation using shaded overlapping regions. There is one region for each variable, all circular in the examples here. The interior and exterior of region x corresponds respectively to the values 1 (true) and 0 (false) for variable x . The shading indicates the value of the operation for each combination of regions, with dark denoting 1 and light 0 (some authors use the opposite convention).

The three Venn diagrams in the figure below represent respectively conjunction $x\wedge y$, disjunction $x\vee y$, and complement $\neg x$.

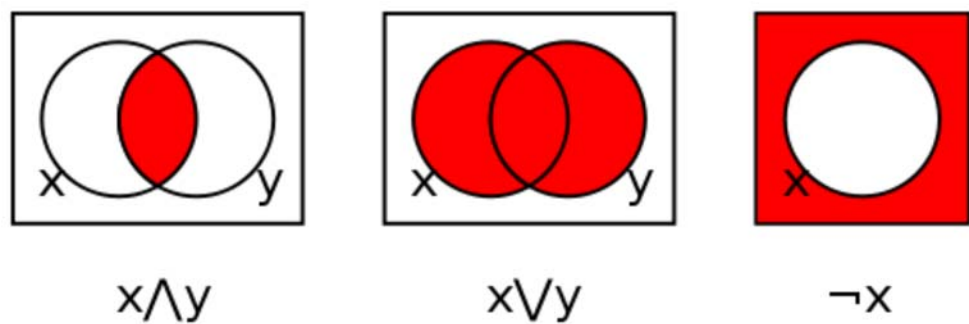


Figure 2. Venn diagrams for conjunction, disjunction, and complement



The second diagram represents disjunction $x \vee y$ by shading those regions that lie inside either or both circles. The third diagram represents complement $\neg x$ by shading the region *not* inside the circle.

While we have not shown the Venn diagrams for the constants 0 and 1, they are trivial, being respectively a white box and a dark box, neither one containing a circle. However we could put a circle for x in those boxes, in which case each would denote a function of one argument, x , which returns the same value independently of x , called a constant function. As far as their outputs are concerned, constants and constant functions are indistinguishable; the difference is that a constant takes no arguments, called a *zeroary* or *nullary* operation, while a constant function takes one argument, which it ignores, and is a *unary* operation.

Venn diagrams are helpful in visualizing laws. The commutativity laws for \wedge and \vee can be seen from the symmetry of the diagrams: a binary operation that was not commutative would not have a symmetric diagram because interchanging x and y would have the effect of reflecting the diagram horizontally and any failure of commutativity would then appear as a failure of symmetry.

Idempotence of \wedge and \vee can be visualized by sliding the two circles together and noting that the shaded area then becomes the whole circle, for both \wedge and \vee .

To see the first absorption law, $x \wedge (x \vee y) = x$, start with the diagram in the middle for $x \vee y$ and note that the portion of the shaded area in common with the x circle is the whole of the x circle. For the second absorption law, $x \vee (x \wedge y) = x$, start with the left diagram for $x \wedge y$ and note that shading the whole of the x circle results in just the x circle being shaded, since the previous shading was inside the x circle.

The double negation law can be seen by complementing the shading in the third diagram for $\neg x$, which shades the x circle.

To visualize the first De Morgan's law, $(\neg x) \wedge (\neg y) = \neg(x \vee y)$, start with the middle diagram for $x \vee y$ and complement its shading so that only the region outside both circles is shaded, which is what the right hand side of the law describes. The result is the same as if we shaded that region which is both outside the x circle *and* outside the y circle, i.e. the conjunction of their exteriors, which is what the left hand side of the law describes.

The second De Morgan's law, $(\neg x) \vee (\neg y) = \neg(x \wedge y)$, works the same way with the two diagrams interchanged.

The first complement law, $x \wedge \neg x = 0$, says that the interior and exterior of the x circle have no overlap. The second complement law, $x \vee \neg x = 1$, says that everything is either inside or outside the x circle.

Digital logic gates

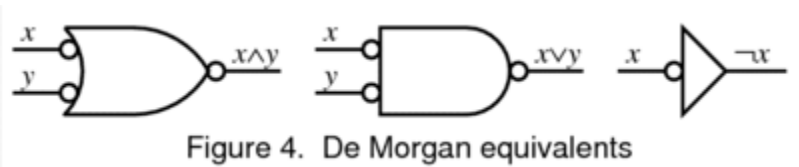
Digital logic is the application of the Boolean algebra of 0 and 1 to electronic hardware consisting of logic gates connected to form a circuit diagram. Each gate implements a Boolean operation, and is depicted schematically by a shape indicating the operation. The shapes associated with the gates for conjunction (AND-gates), disjunction (OR-gates), and complement (inverters) are as follows.^[17]

Figure 3. Logic gates

The lines on the left of each gate represent input wires or *ports*. The value of the input is represented by a voltage on the lead. For so-called "active-high" logic, 0 is represented by a voltage close to zero or "ground", while 1 is represented by a voltage close to the supply voltage; active-low reverses this. The line on the right of each gate represents the output port, which normally follows the same voltage conventions as the input ports.

Complement is implemented with an inverter gate. The triangle denotes the operation that simply copies the input to the output; the small circle on the output denotes the actual inversion complementing the input. The convention of putting such a circle on any port means that the signal passing through this port is complemented on the way through, whether it is an input or output port.

The Duality Principle, or De Morgan's laws, can be understood as asserting that complementing all three ports of an AND gate converts it to an OR gate and vice versa, as shown in Figure 4 below. Complementing both ports of an inverter however leaves the operation unchanged.



More generally one may complement any of the eight subsets of the three ports of either an AND or OR gate. The resulting sixteen possibilities give rise to only eight Boolean operations, namely those with an odd number of 1's in their truth table. There are eight such because the "odd-bit-out" can be either 0 or 1 and can go in any of four positions in the truth table. There being sixteen binary Boolean operations, this must leave eight operations with an even number of 1's in their truth tables. Two of these are the constants 0 and 1 (as binary operations that ignore both their inputs); four are the operations that depend nontrivially on exactly one of their two inputs, namely x , y , $\neg x$, and $\neg y$; and the remaining two are $x \oplus y$ (XOR) and its complement $x \equiv y$.

Boolean algebras

Main article: Boolean algebra (structure)

The term "algebra" denotes both a subject, namely the subject of algebra, and an object, namely an algebraic structure. Whereas the foregoing has addressed the subject of Boolean algebra, this section deals with mathematical objects called Boolean algebras, defined in full generality as any model of the Boolean laws. We begin with a special case of the notion definable without reference to the laws, namely concrete Boolean algebras, and then give the formal definition of the general notion.

Concrete Boolean algebras

A **concrete Boolean algebra** or field of sets is any nonempty set of subsets of a given set X closed under the set operations of union, intersection, and complement relative to X .^[4]



"Boolean algebra," there being no way to rule out the one-element algebra using only equations— $0 \neq 1$ does not count, being a negated equation. Hence modern authors allow the degenerate Boolean algebra and let X be empty.)

Example 1. The power set 2^X of X , consisting of all subsets of X . Here X may be any set: empty, finite, infinite, or even uncountable.

Example 2. The empty set and X . This two-element algebra shows that a concrete Boolean algebra can be finite even when it consists of subsets of an infinite set. It can be seen that every field of subsets of X must contain the empty set and X . Hence no smaller example is possible, other than the degenerate algebra obtained by taking X to be empty so as to make the empty set and X coincide.

Example 3. The set of finite and cofinite sets of integers, where a cofinite set is one omitting only finitely many integers. This is clearly closed under complement, and is closed under union because the union of a cofinite set with any set is cofinite, while the union of two finite sets is finite. Intersection behaves like union with "finite" and "cofinite" interchanged.

Example 4. For a less trivial example of the point made by Example 2, consider a Venn diagram formed by n closed curves partitioning the diagram into 2^n regions, and let X be the (infinite) set of all points in the plane not on any curve but somewhere within the diagram. The interior of each region is thus an infinite subset of X , and every point in X is in exactly one region. Then the set of all 2^{2^n} possible unions of regions (including the empty set obtained as the union of the empty set of regions and X obtained as the union of all 2^n regions) is closed under union, intersection, and complement relative to X and therefore forms a concrete Boolean algebra. Again we have finitely many subsets of an infinite set forming a concrete Boolean algebra, with Example 2 arising as the case $n = 0$ of no curves.

Subsets as bit vectors

A subset Y of X can be identified with an indexed family of bits with index set X , with the bit indexed by $x \in X$ being 1 or 0 according to whether or not $x \in Y$. (This is the so-called characteristic function notion of a subset.) For example, a 32-bit computer word consists of 32 bits indexed by the set $\{0, 1, 2, \dots, 31\}$, with 0 and 31 indexing the low and high order bits respectively. For a smaller example, if $X = \{a, b, c\}$ where a, b, c are viewed as bit positions in that order from left to right, the eight subsets $\{\}, \{c\}, \{b\}, \{b, c\}, \{a\}, \{a, c\}, \{a, b\}$, and $\{a, b, c\}$ of X can be identified with the respective bit vectors 000, 001, 010, 011, 100, 101, 110, and 111. Bit vectors indexed by the set of natural numbers are infinite sequences of bits, while those indexed by the reals in the unit interval $[0, 1]$ are packed too densely to be able to write conventionally but nonetheless form well-defined indexed families (imagine coloring every point of the interval $[0, 1]$ either black or white independently; the black points then form an arbitrary subset of $[0, 1]$).

From this bit vector viewpoint, a concrete Boolean algebra can be defined equivalently as a nonempty set of bit vectors all of the same length (more generally, indexed by the same set) and closed under the bit vector operations of bitwise \wedge , \vee , and \neg , as in $1010 \wedge 0110 = 0010$, $1010 \vee 0110 = 1110$, and $\neg 1010 = 0101$, the bit vector realizations of intersection, union, and complement respectively.

The prototypical Boolean algebra

Main article: two-element Boolean algebra

The laws satisfied by all nondegenerate concrete Boolean algebras coincide with those satisfied by the prototypical Boolean algebra.

This observation is easily proved as follows. Certainly any law satisfied by all concrete Boolean algebras is satisfied by the prototypical one since it is concrete. Conversely any law that fails for some concrete Boolean algebra must have failed at a particular bit position, in which case that position by itself furnishes a one-bit counterexample to that law. Nondegeneracy ensures the existence of at least one bit position because there is only one empty bit vector.

The final goal of the next section can be understood as eliminating "concrete" from the above observation. We shall however reach that goal via the surprisingly stronger observation that, up to isomorphism, all Boolean algebras are concrete.

Boolean algebras: the definition

The Boolean algebras we have seen so far have all been concrete, consisting of bit vectors or equivalently of subsets of some set. Such a Boolean algebra consists of a set and operations on that set which can be *shown* to satisfy the laws of Boolean algebra.

Instead of showing that the Boolean laws are satisfied, we can instead postulate a set X , two binary operations on X , and one unary operation, and *require* that those operations satisfy the laws of Boolean algebra. The elements of X need not be bit vectors or subsets but can be anything at all. This leads to the more general *abstract* definition.

A Boolean algebra is any set with binary operations \wedge and \vee and a unary operation \neg thereon satisfying the Boolean laws.^[18]

For the purposes of this definition it is irrelevant how the operations came to satisfy the laws, whether by fiat or proof. All concrete Boolean algebras satisfy the laws (by proof rather than fiat), whence every concrete Boolean algebra is a Boolean algebra according to our definitions. This axiomatic definition of a Boolean algebra as a set and certain operations satisfying certain laws or axioms *by fiat* is entirely analogous to the abstract definitions of group, ring, field etc. characteristic of modern or abstract algebra.

Given any complete axiomatization of Boolean algebra, such as the axioms for a complemented distributive lattice, a sufficient condition for an algebraic structure of this kind to satisfy all the Boolean laws is that it satisfy just those axioms. The following is therefore an equivalent definition.

A Boolean algebra is a complemented distributive lattice.

The section on axiomatization lists other axiomatizations, any of which can be made the basis of an equivalent definition.

Representable Boolean algebras

Although every concrete Boolean algebra is a Boolean algebra, not every Boolean algebra need be concrete. Let n be a square-free positive integer, one not divisible by the square of an integer, for example 30 but not 12. The operations of greatest common divisor, least common multiple, and division into n (that is, $\neg x = n/x$), can be shown to satisfy all the



However, if we *represent* each divisor of n by the set of its prime factors, we find that this nonconcrete Boolean algebra is isomorphic to the concrete Boolean algebra consisting of all sets of prime factors of n , with union corresponding to least common multiple, intersection to greatest common divisor, and complement to division into n . So this example while not technically concrete is at least "morally" concrete via this representation, called an isomorphism. This example is an instance of the following notion.

A Boolean algebra is called **representable** when it is isomorphic to a concrete Boolean algebra.

The obvious next question is answered positively as follows.

Every Boolean algebra is representable.

That is, up to isomorphism, abstract and concrete Boolean algebras are the same thing. This quite nontrivial result depends on the Boolean prime ideal theorem, a choice principle slightly weaker than the axiom of choice, and is treated in more detail in the article Stone's representation theorem for Boolean algebras. This strong relationship implies a weaker result strengthening the observation in the previous subsection to the following easy consequence of representability.

The laws satisfied by all Boolean algebras coincide with those satisfied by the prototypical Boolean algebra.

It is weaker in the sense that it does not of itself imply representability. Boolean algebras are special here, for example a relation algebra is a Boolean algebra with additional structure but it is not the case that every relation algebra is representable in the sense appropriate to relation algebras.

Axiomatizing Boolean algebra

Main articles: Axiomatization of Boolean algebras and Boolean algebras canonically defined

The above definition of an abstract Boolean algebra as a set and operations satisfying "the" Boolean laws raises the question, what are those laws? A simple-minded answer is "all Boolean laws," which can be defined as all equations that hold for the Boolean algebra of 0 and 1. Since there are infinitely many such laws this is not a terribly satisfactory answer in practice, leading to the next question: does it suffice to require only finitely many laws to hold?

In the case of Boolean algebras the answer is yes. In particular the finitely many equations we have listed above suffice. We say that Boolean algebra is **finitely axiomatizable** or **finitely based**.

Can this list be made shorter yet? Again the answer is yes. To begin with, some of the above laws are implied by some of the others. A sufficient subset of the above laws consists of the pairs of associativity, commutativity, and absorption laws, distributivity of \wedge over \vee (or the other distributivity law—one suffices), and the two complement laws. In fact this is the traditional axiomatization of Boolean algebra as a complemented distributive lattice.

By introducing additional laws not listed above it becomes possible to shorten the list yet further. In 1933, Edward Huntington showed that if the basic operations are taken to be $x \vee y$ and $\neg x$, with $x \wedge y$ considered a derived operation (e.g. via De Morgan's law in the form $x \wedge y = \neg(\neg x \vee \neg y)$), then the equation $\neg(\neg x \vee \neg y) \vee \neg(\neg x \vee y) = x$ along with the two equations expressing associativity and commutativity of \vee completely axiomatized Boolean algebra. When the only basic



Propositional logic

Main article: [Propositional calculus](#)

Propositional logic is a logical system that is intimately connected to Boolean algebra.^[4] Many syntactic concepts of Boolean algebra carry over to propositional logic with only minor changes in notation and terminology, while the semantics of propositional logic are defined via Boolean algebras in a way that the tautologies (theorems) of propositional logic correspond to equational theorems of Boolean algebra.

Syntactically, every Boolean term corresponds to a **propositional formula** of propositional logic. In this translation between Boolean algebra and propositional logic, Boolean variables x, y, \dots become **propositional variables** (or **atoms**) P, Q, \dots , Boolean terms such as $x \vee y$ become propositional formulas $P \vee Q$, 0 becomes *false* or \perp , and 1 becomes *true* or **T**. It is convenient when referring to generic propositions to use Greek letters Φ, Ψ, \dots as metavariables (variables outside the language of propositional calculus, used when talking *about* propositional calculus) to denote propositions.

The semantics of propositional logic rely on **truth assignments**. The essential idea of a truth assignment is that the propositional variables are mapped to elements of a fixed Boolean algebra, and then the **truth value** of a propositional formula using these letters is the element of the Boolean algebra that is obtained by computing the value of the Boolean term corresponding to the formula. In classical semantics, only the two-element Boolean algebra is used, while in Boolean-valued semantics arbitrary Boolean algebras are considered. A **tautology** is a propositional formula that is assigned truth value *1* by every truth assignment of its propositional variables to an arbitrary Boolean algebra (or, equivalently, every truth assignment to the two element Boolean algebra).

These semantics permit a translation between tautologies of propositional logic and equational theorems of Boolean algebra. Every tautology Φ of propositional logic can be expressed as the Boolean equation $\Phi = 1$, which will be a theorem of Boolean algebra. Conversely every theorem $\Phi = \Psi$ of Boolean algebra corresponds to the tautologies $(\Phi \vee \neg \Psi) \wedge (\neg \Phi \vee \Psi)$ and $(\Phi \wedge \Psi) \vee (\neg \Phi \wedge \neg \Psi)$. If \rightarrow is in the language these last tautologies can also be written as $(\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$, or as two separate theorems $\Phi \rightarrow \Psi$ and $\Psi \rightarrow \Phi$; if \equiv is available then the single tautology $\Phi \equiv \Psi$ can be used.

Applications

One motivating application of propositional calculus is the analysis of propositions and deductive arguments in natural language. Whereas the proposition "if $x = 3$ then $x + 1 = 4$ " depends on the meanings of such symbols as $+$ and 1 , the proposition "if $x = 3$ then $x = 3$ " does not; it is true merely by virtue of its structure, and remains true whether " $x = 3$ " is replaced by " $x = 4$ " or "the moon is made of green cheese." The generic or abstract form of this tautology is "if P then P ", or in the language of Boolean algebra, " $P \rightarrow P$ ".

Replacing P by $x = 3$ or any other proposition is called **instantiation** of P by that proposition. The result of instantiating P in an abstract proposition is called an **instance** of the proposition. Thus " $x = 3 \rightarrow x = 3$ " is a tautology by virtue of being an instance of the abstract tautology " $P \rightarrow P$ ". All occurrences of the instantiated variable must be instantiated with the same proposition, to avoid such nonsense as $P \rightarrow x = 3$ or $x = 3 \rightarrow x = 4$.

Propositional calculus restricts attention to abstract propositions, those built up from propositional variables using



(The availability of instantiation as part of the machinery of propositional calculus avoids the need for metavariables within the language of propositional calculus, since ordinary propositional variables can be considered within the language to denote arbitrary propositions. The metavariables themselves are outside the reach of instantiation, not being part of the language of propositional calculus but rather part of the same language for talking about it that this sentence is written in, where we need to be able to distinguish propositional variables and their instantiations as being distinct syntactic entities.)

Deductive systems for propositional logic

An axiomatization of propositional calculus is a set of tautologies called axioms and one or more inference rules for producing new tautologies from old. A *proof* in an axiom system *A* is a finite nonempty sequence of propositions each of which is either an instance of an axiom of *A* or follows by some rule of *A* from propositions appearing earlier in the proof (thereby disallowing circular reasoning). The last proposition is the **theorem** proved by the proof. Every nonempty initial segment of a proof is itself a proof, whence every proposition in a proof is itself a theorem. An axiomatization is **sound** when every theorem is a tautology, and **complete** when every tautology is a theorem.^[19]

Sequent calculus

Main article: Sequent calculus

Propositional calculus is commonly organized as a Hilbert system, whose operations are just those of Boolean algebra and whose theorems are Boolean tautologies, those Boolean terms equal to the Boolean constant 1. Another form is sequent calculus, which has two sorts, propositions as in ordinary propositional calculus, and pairs of lists of propositions called sequents, such as $A \vee B, A \wedge C, \dots \vdash A, B \rightarrow C, \dots$. The two halves of a sequent are called the antecedent and the succedent respectively. The customary metavariable denoting an antecedent or part thereof is Γ , and for a succedent Δ ; thus $\Gamma, A \vdash \Delta$ would denote a sequent whose succedent is a list Δ and whose antecedent is a list Γ with an additional proposition *A* appended after it. The antecedent is interpreted as the conjunction of its propositions, the succedent as the disjunction of its propositions, and the sequent itself as the entailment of the succedent by the antecedent.

Entailment differs from implication in that whereas the latter is a binary *operation* that returns a value in a Boolean algebra, the former is a binary *relation* which either holds or does not hold. In this sense entailment is an *external* form of implication, meaning external to the Boolean algebra, thinking of the reader of the sequent as also being external and interpreting and comparing antecedents and succedents in some Boolean algebra. The natural interpretation of \vdash is as \leq in the partial order of the Boolean algebra defined by $x \leq y$ just when $x \vee y = y$. This ability to mix external implication \vdash and internal implication \rightarrow in the one logic is among the essential differences between sequent calculus and propositional calculus.^[20]

Applications

Boolean algebra as the calculus of two values is fundamental to computer circuits, computer programming, and mathematical logic, and is also used in other areas of mathematics such as set theory and statistics.^[4]

Computers



Today, all modern general purpose computers perform their functions using two-value Boolean logic; that is, their electrical circuits are a physical manifestation of two-value Boolean logic. They achieve this in various ways: as voltages on wires in high-speed circuits and capacitive storage devices, as orientations of a magnetic domain in ferromagnetic storage devices, as holes in punched cards or paper tape, and so on. (Some early computers used decimal circuits or mechanisms instead of two-valued logic circuits.)

Of course, it is possible to code more than two symbols in any given medium. For example, one might use respectively 0, 1, 2, and 3 volts to code a four-symbol alphabet on a wire, or holes of different sizes in a punched card. In practice, the tight constraints of high speed, small size, and low power combine to make noise a major factor. This makes it hard to distinguish between symbols when there are several possible symbols that could occur at a single site. Rather than attempting to distinguish between four voltages on one wire, digital designers have settled on two voltages per wire, high and low.

Computers use two-value Boolean circuits for the above reasons. The most common computer architectures use ordered sequences of Boolean values, called bits, of 32 or 64 values, e.g. 01101000110101100101010101001011. When programming in machine code, assembly language, and certain other programming languages, programmers work with the low-level digital structure of the data registers. These registers operate on voltages, where zero volts represents Boolean 0, and a reference voltage (often +5V, +3.3V, +1.8V) represents Boolean 1. Such languages support both numeric operations and logical operations. In this context, "numeric" means that the computer treats sequences of bits as binary numbers (base two numbers) and executes arithmetic operations like add, subtract, multiply, or divide. "Logical" refers to the Boolean logical operations of disjunction, conjunction, and negation between two sequences of bits, in which each bit in one sequence is simply compared to its counterpart in the other sequence. Programmers therefore have the option of working in and applying the rules of either numeric algebra or Boolean algebra as needed. A core differentiating feature between these families of operations is the existence of the carry operation in the first but not the second.

Two-valued logic

Other areas where two values is a good choice are the law and mathematics. In everyday relaxed conversation, nuanced or complex answers such as "maybe" or "only on the weekend" are acceptable. In more focused situations such as a court of law or theorem-based mathematics however it is deemed advantageous to frame questions so as to admit a simple yes-or-no answer—is the defendant guilty or not guilty, is the proposition true or false—and to disallow any other answer. However much of a straitjacket this might prove in practice for the respondent, the principle of the simple yes-no question has become a central feature of both judicial and mathematical logic, making two-valued logic deserving of organization and study in its own right.

A central concept of set theory is membership. Now an organization may permit multiple degrees of membership, such as novice, associate, and full. With sets however an element is either in or out. The candidates for membership in a set work just like the wires in a digital computer: each candidate is either a member or a nonmember, just as each wire is either high or low.

Algebra being a fundamental tool in any area amenable to mathematical treatment, these considerations combine to make the algebra of two values of fundamental importance to computer hardware, mathematical logic, and set theory.



and disjunction (OR) is defined via De Morgan's law. Interpreting these values as logical truth values yields a multi-valued logic, which forms the basis for fuzzy logic and probabilistic logic. In these interpretations, a value is interpreted as the "degree" of truth – to what extent a proposition is true, or the probability that the proposition is true.

Boolean operations

The original application for Boolean operations was mathematical logic, where it combines the truth values, true or false, of individual formulas.

Natural languages such as English have words for several Boolean operations, in particular conjunction (*and*), disjunction (*or*), negation (*not*), and implication (*implies*). *But not* is synonymous with *and not*. When used to combine situational assertions such as "the block is on the table" and "cats drink milk," which naively are either true or false, the meanings of these logical connectives often have the meaning of their logical counterparts. However, with descriptions of behavior such as "Jim walked through the door", one starts to notice differences such as failure of commutativity, for example the conjunction of "Jim opened the door" with "Jim walked through the door" in that order is not equivalent to their conjunction in the other order, since *and* usually means *and then* in such cases. Questions can be similar: the order "Is the sky blue, and why is the sky blue?" makes more sense than the reverse order. Conjunctive commands about behavior are like behavioral assertions, as in *get dressed and go to school*. Disjunctive commands such *love me or leave me* or *fish or cut bait* tend to be asymmetric via the implication that one alternative is less preferable. Conjoined nouns such as *tea and milk* generally describe aggregation as with set union while *tea or milk* is a choice. However context can reverse these senses, as in *your choices are coffee and tea* which usually means the same as *your choices are coffee or tea* (alternatives). Double negation as in "I don't not like milk" rarely means literally "I do like milk" but rather conveys some sort of hedging, as though to imply that there is a third possibility. "Not not P" can be loosely interpreted as "surely P", and although *P* necessarily implies "not not *P*" the converse is suspect in English, much as with intuitionistic logic. In view of the highly idiosyncratic usage of conjunctions in natural languages, Boolean algebra cannot be considered a reliable framework for interpreting them.

Boolean operations are used in digital logic to combine the bits carried on individual wires, thereby interpreting them over $\{0,1\}$. When a vector of n identical binary gates are used to combine two bit vectors each of n bits, the individual bit operations can be understood collectively as a single operation on values from a Boolean algebra with 2^n elements.

Naive set theory interprets Boolean operations as acting on subsets of a given set X . As we saw earlier this behavior exactly parallels the coordinate-wise combinations of bit vectors, with the union of two sets corresponding to the disjunction of two bit vectors and so on.

The 256-element free Boolean algebra on three generators is deployed in computer displays based on raster graphics, which use bit blit to manipulate whole regions consisting of pixels, relying on Boolean operations to specify how the source region should be combined with the destination, typically with the help of a third region called the mask. Modern video cards offer all $2^{2^3} = 256$ ternary operations for this purpose, with the choice of operation being a one-byte (8-bit) parameter. The constants SRC = 0xaa or 10101010, DST = 0xcc or 11001100, and MSK = 0xf0 or 11110000 allow Boolean operations such as (SRC^DST)&MSK (meaning XOR the source and destination and then AND the result with the mask) to be written directly as a constant denoting a byte calculated at compile time, 0x60 in the (SRC^DST)&MSK example, 0x66 if just SRC^DST, etc. At run time the video card interprets the byte as the raster operation indicated by the original



Solid modeling systems for computer aided design offer a variety of methods for building objects from other objects, combination by Boolean operations being one of them. In this method the space in which objects exist is understood as a set S of voxels (the three-dimensional analogue of pixels in two-dimensional graphics) and shapes are defined as subsets of S , allowing objects to be combined as sets via union, intersection, etc. One obvious use is in building a complex shape from simple shapes simply as the union of the latter. Another use is in sculpting understood as removal of material: any grinding, milling, routing, or drilling operation that can be performed with physical machinery on physical materials can be simulated on the computer with the Boolean operation $x \wedge \neg y$ or $x - y$, which in set theory is set difference, remove the elements of y from those of x . Thus given two shapes one to be machined and the other the material to be removed, the result of machining the former to remove the latter is described simply as their set difference.

Boolean searches

Search engine queries also employ Boolean logic. For this application, each web page on the Internet may be considered to be an "element" of a "set". The following examples use a syntax supported by Google.^[21]

- Doublequotes are used to combine whitespace-separated words into a single search term.^[22]
 - Whitespace is used to specify logical AND, as it is the default operator for joining search terms:
- ```
"Search term 1" "Search term 2"
```

- The OR keyword is used for logical OR:
- ```
"Search term 1" OR "Search term 2"
```

- A prefixed minus sign is used for logical NOT:
- ```
"Search term 1" -"Search term 2"
```

**See also**

---

- Binary number
- Boolean algebra (structure)
- Boolean algebras canonically defined
- Boolean differential calculus
- Booleo
- Heyting algebra
- Intuitionistic logic
- List of Boolean algebra topics
- Logic design
- Propositional calculus
- Relation algebra
- Three-valued logic
- Vector logic





ISBN 978-1-59102-089-9.

2. ↑ "The name Boolean algebra (or Boolean 'algebras') for the calculus originated by Boole, extended by Schröder, and perfected by Whitehead seems to have been first suggested by Sheffer, in 1913." E. V. Huntington, "New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's *Principia mathematica* (<http://www.ams.org/journals/tran/1933-035-01/S0002-9947-1933-1501684-X/S0002-9947-1933-1501684-X.pdf>)", in *Trans. Amer. Math. Soc.* **35** (1933), 274-304; footnote, page 278.
3. ↑ Peirce, Charles S. (1931). *Collected Papers* (<https://books.google.com/books?id=3JJgOkGmnjEC&pg=RA1-PA13>). 3. Harvard University Press. p. 13. ISBN 978-0-674-13801-8.
4. 1 2 3 4 5 6 Givant, Steven; Halmos, Paul (2009). *Introduction to Boolean Algebras*. Undergraduate Texts in Mathematics, Springer. ISBN 978-0-387-40293-2.
5. 1 2 3 J. Michael Dunn; Gary M. Hardegree (2001). *Algebraic methods in philosophical logic* (<https://books.google.com/books?id=-AokWhbILUIC&pg=PA2>). Oxford University Press US. p. 2. ISBN 978-0-19-853192-0.
6. ↑ Norman Balabanian; Bradley Carlson (2001). *Digital logic design principles*. John Wiley. pp. 39–40. ISBN 978-0-471-29351-4. , online sample (<http://www.wiley.com/college/engin/balabanian293512/pdf/ch02.pdf>)
7. ↑ Rajaraman & Radhakrishnan. *Introduction To Digital Computer Design* (<https://books.google.com/books?id=-8MvcOgsSjcC&pg=PA65>). PHI Learning Pvt. Ltd. p. 65. ISBN 978-81-203-3409-0.
8. ↑ John A. Camara (2010). *Electrical and Electronics Reference Manual for the Electrical and Computer PE Exam* (<https://books.google.com/books?id=rHWHHeU0jfsC&pg=SA41-PA3>). www.ppi2pass.com. p. 41. ISBN 978-1-59126-166-7.
9. ↑ Shin-ichi Minato, Saburo Muroga (2007). "Binary Decision Diagrams". In Wai-Kai Chen. *The VLSI handbook* (2nd ed.). CRC Press. ISBN 978-0-8493-4199-1. chapter 29.
10. ↑ Alan Parkes (2002). *Introduction to languages, machines and logic: computable languages, abstract machines and formal logic* (<https://books.google.com/books?id=sUQXKy8KPcQC&pg=PA276>). Springer. p. 276. ISBN 978-1-85233-464-2.
11. ↑ Jon Barwise; John Etchemendy; Gerard Allwein; Dave Barker-Plummer; Albert Liu (1999). *Language, proof, and logic*. CSLI Publications. ISBN 978-1-889119-08-3.
12. ↑ Ben Goertzel (1994). *Chaotic logic: language, thought, and reality from the perspective of complex systems science* (<https://books.google.com/books?id=zVOWoXDunp8C&pg=PA48>). Springer. p. 48. ISBN 978-0-306-44690-0.
13. ↑ Halmos, Paul (1963). *Lectures on Boolean Algebras*. van Nostrand.
14. ↑ O'Regan, Gerard (2008). *A brief history of computing* (<https://books.google.com/books?id=081H96F1enMC&pg=PA33>). Springer. p. 33. ISBN 978-1-84800-083-4.
15. ↑ Steven R. Givant; Paul Richard Halmos (2009). *Introduction to Boolean algebras* (<https://books.google.com/books?id=ORILyf8sF2sC&pg=PA22>). Springer. pp. 21–22. ISBN 978-0-387-40293-2.
16. ↑ Venn, John (July 1880). "I. On the Diagrammatic and Mechanical Representation of Propositions and Reasonings" ([https://www.cis.upenn.edu/~bhusnur4/cit592\\_fall2014/venn%20diagrams.pdf](https://www.cis.upenn.edu/~bhusnur4/cit592_fall2014/venn%20diagrams.pdf)) (PDF). *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. 5. Taylor & Francis. **10** (59): 1–18. doi:10.1080/14786448008626877 (<http://doi.org/10.1080%2F14786448008626877>). Archived ([https://web.archive.org/web/20170516204620/https://www.cis.upenn.edu/~bhusnur4/cit592\\_fall2014/venn%20diagrams.pdf](https://web.archive.org/web/20170516204620/https://www.cis.upenn.edu/~bhusnur4/cit592_fall2014/venn%20diagrams.pdf)) (PDF) from the original on 2017-05-16. (<http://www.tandfonline.com/doi/abs/10.1080/14786448008626877>) (<https://books.google.com/books?id=k68vAQAAIAAJ&pg=PA1>)
17. ↑ Shannon, Claude (1949). "The Synthesis of Two-Terminal Switching Circuits". *Bell System Technical Journal*. **28**: 59–98. doi:10.1002/j.1538-7305.1949.tb03624.x (<http://doi.org/10.1002%2Fj.1538-7305.1949.tb03624.x>).
18. ↑ Koppelberg, Sabine (1989). "General Theory of Boolean Algebras". *Handbook of Boolean Algebras, Vol. 1* (ed. J. Donald Monk with Robert Bonnet). Amsterdam: North Holland. ISBN 978-0-444-70261-6.



/Proofs%2BTypes.html). Cambridge University Press (Cambridge Tracts in Theoretical Computer Science, 7). ISBN 0-521-37181-3.

21. ↑ Not all search engines support the same query syntax. Additionally, some organizations (such as Google) provide "specialized" search engines that support alternate or extended syntax. (See e.g., [Syntax cheatsheet](https://www.google.com/help/cheatsheet.html) (<https://www.google.com/help/cheatsheet.html>), [Google codesearch supports regular expressions](https://www.google.com/intl/en/help/faq_codesearch.html#regexp) ([https://www.google.com/intl/en/help/faq\\_codesearch.html#regexp](https://www.google.com/intl/en/help/faq_codesearch.html#regexp))).
22. ↑ Doublequote-delimited search terms are called "exact phrase" searches in the Google documentation.

## General

Mano, Morris; Ciletti, Michael D. (2013). *Digital Design*. Pearson. ISBN 978-0-13-277420-8.

## Further reading

- J. Eldon Whitesitt (1995). *Boolean algebra and its applications*. Courier Dover Publications. ISBN 978-0-486-68483-3. Suitable introduction for students in applied fields.
- Dwinger, Philip (1971). *Introduction to Boolean algebras*. Würzburg: Physica Verlag.
- Sikorski, Roman (1969). *Boolean Algebras* (3/e ed.). Berlin: Springer-Verlag. ISBN 978-0-387-04469-9.
- Bocheński, Józef Maria (1959). *A Précis of Mathematical Logic*. Translated from the French and German editions by Otto Bird. Dordrecht, South Holland: D. Reidel.

## Historical perspective

- George Boole (1848). "The Calculus of Logic, (<http://www.maths.tcd.ie/pub/HistMath/People/Boole/CalcLogic/CalcLogic.html>)" *Cambridge and Dublin Mathematical Journal III*: 183–98.
- Theodore Hailperin (1986). *Boole's logic and probability: a critical exposition from the standpoint of contemporary algebra, logic, and probability theory* (2nd ed.). Elsevier. ISBN 978-0-444-87952-3.
- Dov M. Gabbay, John Woods, ed. (2004). *The rise of modern logic: from Leibniz to Frege*. Handbook of the History of Logic. 3. Elsevier. ISBN 978-0-444-51611-4. , several relevant chapters by Hailperin, Valencia, and Grattan-Guinness
- Calixto Badesa (2004). *The birth of model theory: Löwenheim's theorem in the frame of the theory of relatives*. Princeton University Press. ISBN 978-0-691-05853-5. , chapter 1, "Algebra of Classes and Propositional Calculus"
- Burris, Stanley, 2009. The Algebra of Logic Tradition (<http://plato.stanford.edu/entries/algebra-logic-tradition/>). Stanford Encyclopedia of Philosophy.
- Radomir S. Stankovic; Jaakko Astola (2011). *From Boolean Logic to Switching Circuits and Automata: Towards Modern Information Technology* (<https://books.google.com/books?id=uagvEc2jGTIC>). Springer. ISBN 978-3-642-11681-0.

## External links

- Boolean Algebra ([http://www.allaboutcircuits.com/vol\\_4/chpt\\_7/1.html](http://www.allaboutcircuits.com/vol_4/chpt_7/1.html)) chapter on All About Circuits
- How Stuff Works – Boolean Logic (<http://computer.howstuffworks.com/boolean.htm>)
- Science and Technology - Boolean Algebra (<http://osscience.info/mathematics/boolean-algebra-2/>) contains a list and proof of Boolean theorems and laws.



|                      |                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Theory</b>        | <u>Computer architecture</u> · <u>Digital signal</u> ( <u>Digital signal processing</u> ) · <u>Circuit minimization</u> · <u>Switching circuit theory</u>                                                                                                                                                                                                     |
| <b>Design</b>        | <u>Logic synthesis</u> · <u>Place and route</u> ( <u>Placement</u> · <u>Routing</u> ) · <u>Register-transfer level</u> ( <u>Hardware description language</u> · <u>High-level synthesis</u> ) · <u>Formal equivalence checking</u> · <u>Synchronous logic</u> · <u>Asynchronous logic</u> · <u>Finite-state machine</u> ( <u>Hierarchical state machine</u> ) |
| <b>Applications</b>  | <u>Computer hardware</u> ( <u>Hardware acceleration</u> ) · <u>Digital audio</u> ( <u>radio</u> ) · <u>Digital photography</u> · <u>Digital telephone</u> · <u>Digital video</u> ( <u>cinema</u> · <u>television</u> ) · <u>Electronic literature</u>                                                                                                         |
| <b>Design issues</b> | <u>Metastability</u> · <u>Runt pulse</u>                                                                                                                                                                                                                                                                                                                      |

## Major fields of computer science

Note: This template roughly follows the 2012 ACM Computing Classification System.

|                                      |                                                                                                                                                                                                                                                                                                                    |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Hardware</b>                      | <u>Printed circuit board</u> · <u>Peripheral</u> · <u>Integrated circuit</u> · <u>Very-large-scale integration</u> · <u>Systems on Chip (SoCs)</u> · <u>Energy consumption</u> ( <u>Green computing</u> ) · <u>Electronic design automation</u> · <u>Hardware acceleration</u>                                     |
| <b>Computer systems organization</b> | <u>Computer architecture</u> · <u>Embedded system</u> · <u>Real-time computing</u> · <u>Dependability</u>                                                                                                                                                                                                          |
| <b>Networks</b>                      | <u>Network architecture</u> · <u>Network protocol</u> · <u>Network components</u> · <u>Network scheduler</u> · <u>Network performance evaluation</u> · <u>Network service</u>                                                                                                                                      |
| <b>Software organization</b>         | <u>Interpreter</u> · <u>Middleware</u> · <u>Virtual machine</u> · <u>Operating system</u> · <u>Software quality</u>                                                                                                                                                                                                |
| <b>Software notations and tools</b>  | <u>Programming paradigm</u> · <u>Programming language</u> · <u>Compiler</u> · <u>Domain-specific language</u> · <u>Modeling language</u> · <u>Software framework</u> · <u>Integrated development environment</u> · <u>Software configuration management</u> · <u>Software library</u> · <u>Software repository</u> |
| <b>Software development</b>          | <u>Software development process</u> · <u>Requirements analysis</u> · <u>Software design</u> · <u>Software construction</u> · <u>Software deployment</u> · <u>Software maintenance</u> · <u>Programming team</u> · <u>Open-source model</u>                                                                         |
| <b>Theory of computation</b>         | <u>Model of computation</u> · <u>Formal language</u> · <u>Automata theory</u> · <u>Computational complexity theory</u> · <u>Logic</u> · <u>Semantics</u>                                                                                                                                                           |

**Mathematics  
of computing**

[Discrete mathematics](#) · [Probability](#) · [Statistics](#) · [Mathematical software](#)  
· [Information theory](#) · [Mathematical analysis](#) · [Numerical analysis](#)

**Information  
systems**

[Database management system](#) · [Information storage systems](#) ·  
[Enterprise information system](#) · [Social information systems](#) ·  
[Geographic information system](#) · [Decision support system](#) ·  
[Process control system](#) · [Multimedia information system](#) · [Data mining](#)  
· [Digital library](#) · [Computing platform](#) · [Digital marketing](#) ·  
[World Wide Web](#) · [Information retrieval](#)

**Security**

[Cryptography](#) · [Formal methods](#) · [Security services](#) ·  
[Intrusion detection system](#) · [Hardware security](#) · [Network security](#) ·  
[Information security](#) · [Application security](#)

**Human–computer  
interaction**

[Interaction design](#) · [Social computing](#) · [Ubiquitous computing](#) ·  
[Visualization](#) · [Accessibility](#)

**Concurrency**

[Concurrent computing](#) · [Parallel computing](#) · [Distributed computing](#) ·  
[Multithreading](#) · [Multiprocessing](#)

**Artificial  
intelligence**

[Natural language processing](#) ·  
[Knowledge representation and reasoning](#) · [Computer vision](#) ·  
[Automated planning and scheduling](#) · [Search methodology](#) ·  
[Control method](#) · [Philosophy of artificial intelligence](#) ·  
[Distributed artificial intelligence](#)

**Machine learning**

[Supervised learning](#) · [Unsupervised learning](#) · [Reinforcement learning](#)  
· [Multi-task learning](#) · [Cross-validation](#)

**Graphics**

[Animation](#) · [Rendering](#) · [Image manipulation](#) ·  
[Graphics processing unit](#) · [Mixed reality](#) · [Virtual reality](#) ·  
[Image compression](#) · [Solid modeling](#)

**Applied  
computing**

[E-commerce](#) · [Enterprise software](#) · [Computational mathematics](#) ·  
[Computational physics](#) · [Computational chemistry](#) ·  
[Computational biology](#) · [Computational social science](#) ·  
[Computational engineering](#) · [Computational healthcare](#) · [Digital art](#) ·  
[Electronic publishing](#) · [Cyberwarfare](#) · [Electronic voting](#) · [Video games](#)  
· [Word processing](#) · [Operations research](#) · [Educational technology](#) ·  
[Document management](#)





[Outline](#) · [Topic lists](#)





|          |                           |                                                                                                                                                                                                                               |
|----------|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Branches | <b><u>Arithmetic</u></b>  | <u>Number theory</u>                                                                                                                                                                                                          |
|          | <b><u>Algebra</u></b>     | <u>Elementary</u> · <u>Linear</u> · <u>Multilinear</u> · <u>Abstract</u> · <u>Combinatorics</u> · <u>Group theory</u> · <u>Representation theory</u> · <u>Lie theory</u>                                                      |
|          | <b><u>Calculus</u></b>    | <u>Analysis</u> · <u>Differential equations</u> / <u>Dynamical systems</u> · <u>Numerical analysis</u> · <u>Optimization</u> · <u>Functional analysis</u>                                                                     |
|          | <b><u>Geometry</u></b>    | <u>Discrete</u> · <u>Algebraic</u> · <u>Analytic</u> · <u>Differential</u> · <u>Finite</u> · <u>Topology</u> · <u>Trigonometry</u>                                                                                            |
|          | <b><u>Foundations</u></b> | <u>Philosophy of mathematics</u> · <u>Mathematical logic</u> · <u>Set theory</u> · <u>Category theory</u>                                                                                                                     |
|          | <b><u>Applied</u></b>     | <u>Mathematical physics</u> · <u>Probability</u> · <u>Mathematical statistics</u> · <u>Statistics</u> · <u>Game theory</u> · <u>Information theory</u> · <u>Computer science</u> · <u>Computation</u> · <u>Control theory</u> |
|          | <b><u>Others</u></b>      | <u>History of mathematics</u> · <u>Recreational mathematics</u> · <u>Mathematics and art</u> · <u>Mathematics education</u> · <u>Order theory</u> · <u>Graph theory</u>                                                       |

**Divisions**   Pure · Applied · Discrete · Computational

**Category** · **Portal** · **Commons** · **WikiProject**

## **Mathematical logic**

|                                                 |                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General                                         | <u>Formal language</u> · <u>Formation rule</u> · <u>Formal proof</u> · <u>Formal semantics</u> · <u>Well-formed formula</u> · <u>Set</u> · <u>Element</u> · <u>Class</u> · <u>Classical logic</u> · <u>Axiom</u> · <u>Rule of inference</u> · <u>Relation</u> · <u>Theorem</u> · <u>Logical consequence</u> · <u>Type theory</u> · <u>Symbol</u> · <u>Syntax</u> · <u>Theory</u> |
|                                                 | <u>Formal system</u> · <u>Deductive system</u> · <u>Axiomatic system</u> · <u>Hilbert style systems</u> · <u>Natural deduction</u> · <u>Sequent calculus</u>                                                                                                                                                                                                                     |
| Systems                                         | <u>Proposition</u> · <u>Inference</u> · <u>Argument</u> · <u>Validity</u> · <u>Cogency</u> · <u>Syllogism</u> · <u>Square of opposition</u> · <u>Venn diagram</u>                                                                                                                                                                                                                |
| <u>Traditional logic</u>                        | <u>Boolean functions</u> · <u>Propositional calculus</u> · <u>Propositional formula</u> · <u>Logical connectives</u> · <u>Truth tables</u> · <u>Many-valued logic</u>                                                                                                                                                                                                            |
| <u>Propositional calculus and Boolean logic</u> | <u>First-order</u> · <u>Quantifiers</u> · <u>Predicate</u> · <u>Second-order</u> · <u>Monadic predicate calculus</u>                                                                                                                                                                                                                                                             |
| <u>Predicate logic</u>                          |                                                                                                                                                                                                                                                                                                                                                                                  |

|                                                                                                            |                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                      |  |                                                                                    |
|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|------------------------------------------------------------------------------------|
|                             | Wikipedia                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                      |  |  |
| <b><u>Set theory</u></b>                                                                                   | <a href="#">Recursive set</a> · <a href="#">Domain</a> · <a href="#">Codomain</a> · <a href="#">Image</a> · <a href="#">Map</a> · <a href="#">Function</a> · <a href="#">Relation</a> · <a href="#">Ordered pair</a>                                                                                                                                         |                                                                                                                                                                                                                                                      |  |                                                                                    |
|                                                                                                            | <a href="#">Foundations of mathematics</a> · <a href="#">Zermelo–Fraenkel set theory</a> · <a href="#">Axiom of choice</a> · <a href="#">General set theory</a> · <a href="#">Kripke–Platek set theory</a> · <a href="#">Von Neumann–Bernays–Gödel set theory</a> · <a href="#">Morse–Kelley set theory</a> · <a href="#">Tarski–Grothendieck set theory</a> |                                                                                                                                                                                                                                                      |  |                                                                                    |
| <b><u>Model theory</u></b>                                                                                 | <a href="#">Model</a> · <a href="#">Interpretation</a> · <a href="#">Non-standard model</a> · <a href="#">Finite model theory</a> · <a href="#">Truth value</a> · <a href="#">Validity</a>                                                                                                                                                                   |                                                                                                                                                                                                                                                      |  |                                                                                    |
| <b><u>Proof theory</u></b>                                                                                 | <a href="#">Formal proof</a> · <a href="#">Deductive system</a> · <a href="#">Formal system</a> · <a href="#">Theorem</a> · <a href="#">Logical consequence</a> · <a href="#">Rule of inference</a> · <a href="#">Syntax</a>                                                                                                                                 |                                                                                                                                                                                                                                                      |  |                                                                                    |
| <b><u>Computability theory</u></b>                                                                         | <a href="#">Recursion</a> · <a href="#">Recursive set</a> · <a href="#">Recursively enumerable set</a> · <a href="#">Decision problem</a> · <a href="#">Church–Turing thesis</a> · <a href="#">Computable function</a> · <a href="#">Primitive recursive function</a>                                                                                        |                                                                                                                                                                                                                                                      |  |                                                                                    |
| <b>Authority control</b>  |                                                                                                                                                                                                                                                                                                                                                              | <a href="#">GND: 4146280-4</a> ( <a href="https://d-nb.info/gnd/4146280-4">https://d-nb.info/gnd/4146280-4</a> ) · <a href="#">NDL: 00560863</a> ( <a href="https://id.ndl.go.jp/auth/ndlna/00560863">https://id.ndl.go.jp/auth/ndlna/00560863</a> ) |  |                                                                                    |

This article is issued from Wikipedia ([https://en.wikipedia.org/wiki/Boolean\\_algebra?oldid=862632395](https://en.wikipedia.org/wiki/Boolean_algebra?oldid=862632395)). The text is licensed under [Creative Commons - Attribution - Sharealike](http://creativecommons.org/licenses/by-sa/4.0/) (<http://creativecommons.org/licenses/by-sa/4.0/>). Additional terms may apply for the media files.