from Boolean Algebra to Unified Algebra

Eric C. R. Hehner
University of Toronto

Abstract

Boolean algebra is simpler than number algebra, with applications in programming, circuit design, law, specifications, mathematical proof, and reasoning in any domain. So why is number algebra taught in primary school and used routinely by scientists, engineers, economists, and the general public, while boolean algebra is not taught until university, and not routinely used by anyone? A large part of the answer may be in the terminology of logic, in the symbols used, and in the explanations of boolean algebra found in textbooks. The subject has not yet freed itself from its history and philosophy. This paper points out the some of problems delaying the acceptance and use of boolean algebra, and suggests some solutions.

Introduction

This paper is about the symbols and notations of boolean algebra, and about the way the subject is explained. It is about education, and about putting boolean algebra into general use and practice. To make the scope clear, by “boolean algebra” I mean the usual algebra whose expressions are of type boolean. I mean to include propositional logic and predicate calculus. I shall say “boolean algebra”, “boolean calculus”, or “logic” interchangeably, and call the expressions of this algebra “boolean expressions”. Analogously, I say “number algebra”, “number calculus”, or “arithmetic” interchangeably, and call the expressions of that algebra “number expressions”.

Boolean algebra is the basic algebra for much of computer science. Other applications include digital circuit design, law, reasoning about any subject, and any kind of specifications, as well as providing a foundation for all of mathematics. Boolean algebra is inherently simpler than number algebra. There are only two boolean values and a few boolean operators, and they can be explained by a small table. There are infinitely many number values and number operators, and even the simplest, counting, is inductively defined. So why is number algebra taught in primary school, and boolean algebra in university? Why isn't boolean algebra better known, better accepted, and better used?

One reason may be that, although boolean algebra is just as useful as number algebra, it isn't as necessary. Informal methods of reckoning quantity became intolerable several thousand years ago, but we still get along with informal methods of specification, design, and reasoning. Another

reason may be just an accident of educational history, and still another may be our continuing mistreatment of boolean algebra.

Historical Perspective

To start to answer these questions, I'm going to look briefly at the history of number algebra. Long after the invention of numbers and arithmetic, quantitative reasoning was still a matter of trial and error, and still conducted in natural language. If a man died leaving his 3 goats and 20 chickens to be divided equally between his 2 sons, and it was agreed that a goat is worth 8 chickens, the solution was determined by iterative approximations, probably using the goats and chickens themselves in the calculation. The arithmetic needed for verification was well understood long before the algebra needed to find a solution.

The advent of algebra provided a more effective way of finding solutions to such problems, but it was a difficult step up in abstraction. The step from constants to variables is as large as the step from chickens to numbers. In English 500 years ago, constants were called "nombers denominate" [concrete numbers], and variables were called "nombers abstracte". Each step in an abstract calculation was accompanied by a concrete justification. For example, we have the Commutative Law [0]:

> When the chekyns of two gentle menne are counted, we may count first the chekyns of the gentylman having fewer chekyns, and after the chekyns of the gentylman having the greater portion. If the nomber of the greater portion be counted first, and then that of the lesser portion, the denomination so determined shall be the same.

This version of the Commutative Law includes an unnecessary case analysis, and it has missed a case: when the two gentlemen have the same number of chickens, it does not say whether the order matters. The Associative Law [0]:

> When thynges to be counted are divided in two partes, and lately are found moare thynges to be counted in the same generall quantitie, it matters not whether the thynges lately added be counted together with the lesser parte or with the greater parte, or that there are severalle partes and the thynges lately added be counted together with any one of them.

One of the simplest, most general laws, sometimes called "substitution of equals for equals",

$$x=y \Rightarrow fx=fy$$

seems to have been discovered a little at a time. Here is one special case [14]:

> "In the firste there appeareth 2 nombers, that is $14x + 15y$ equalle to one nomber, whiche is $71y$. But if you marke them well, you maie see one denomination, on bothe sides of the equation, which never ought to stand. Wherfore abating [subtracting] the lesser, that is $15y$ out of bothe the nombers, there will remain $14x = 56y$ that is, by reduction,

> $1x = 4y$.
>
> Scholar. I see, you abate $15y$ from them bothe. And then are thei equalle still, seyng thei wer equalle before. According to the thirde common sentence, in the patthewaie: If you abate even [equal] portions, from thynges that bee equalle, the partes that remain shall be equall also.
>
> Master. You doe well remember the firste grounds of this arte."

And then, a paragraph later, another special case:

> "If you adde equalle portions, to thynges that bee equalle, what so amounteth of them shall be equalle."

As you can imagine, the distance from $2x + 3 = 3x + 2$ to $x=1$ was likely to be several pages. The reason for all the discussion in between formulas was that algebra was not yet fully trusted. Algebra replaces meaning with symbol manipulation; the loss of meaning is not easy to accept. The author had to constantly reassure those readers who had not yet freed themselves from thinking about the objects represented by numbers and variables. Those who were skilled in the art of informal reasoning about quantity were convinced that thinking about the objects helps to calculate correctly, because that is how they did it. As with any technological advance, those who are most skilled in the old way are the most reluctant to see it replaced by the new.

Today, of course, we expect a quantitative calculation to be conducted entirely in algebra, without reference to thynges. Although we justify each step in a calculation by reference to an algebraic law, we do not have to continually justify the laws. We can go farther, faster, more succinctly, and with much greater certainty. In a typical modern proof we see lines like

$$\lambda^r a^r=(bab^{-1})^r=ba^rb^{-1}=a^r$$

$$b^r=\lambda^r b^r=(\lambda b)^r=(a^{-1}ba)^r=a^{-1}b^ra$$

$$(a_1^{-1}b_1)^2=a_1^{-1}b_1a_1^{-1}b_1=a_1^{-1}(b_1a_1^{-1})b_1=a_1^{-1}(\mu a_1^{-1}b_1)b_1=\mu a_1^{-2}b_1^2$$

$$(a_1^{-1}b_1)^r=\mu^{1+2+\ldots+(r-1)}a_1^{-r}b_1^r=\mu^{1+2+\ldots+(r-1)}=\mu^{r(r-1)/2}$$

These lines were taken from a proof of Wedderburn's Theorem (a finite division ring is a commutative field) in [10] (the text used when I studied algebra). Before we start to feel pleased with ourselves at the improvement, let me point out that there is another kind of calculation, a boolean calculation, occurring in the English text between the formulas. In the example proof [10] we find the words "consequently", "implying", "there is/are", "however", "thus", "hence", "since", "forces", "if...then", "in consequence of which", "from which we get", "whence", "would imply", "contrary to", "so that", "contradicting"; all these words suggest boolean operators. We also find bookkeeping sentences like "We first remark ...", "We must now rule out the case ..."; these suggest the structure of a boolean expression. It will be quite a large expression, perhaps taking an entire page. If written in the usual unformatted fashion of proofs in current algebra texts, it will be quite unreadable. The same problem occurs with computer

programs, which can be thousands of pages long; to make them readable they must be carefully formatted, with indentation to indicate logical structure. We will have to do likewise with proofs.

A formal proof is a boolean calculation using boolean algebra; when we learn to use it well, it will enable us to go farther, faster, more succinctly, and with much greater certainty. But there is a great resistance in the mathematical community to formal proof, especially from those who are most expert at informal proof. They complain that formal proof loses meaning, replacing it with symbol manipulation. The current state of boolean algebra, not as an object of study but as a tool for use, is very much the same as number algebra was 5 centuries ago.

Boolean Calculation

Given an expression, it is often useful to find an equivalent but simpler expression. For example, in number algebra

	$x\times(z+1) - y\times(z-1) - z\times(x-y)$	distribute
$=$	$(x\times z + x\times 1) - (y\times z - y\times 1) - (z\times x - z\times y)$	unity and double negation
$=$	$x\times z + x - y\times z + y - z\times x + z\times y$	symmetry and associativity
$=$	$x + y + (x\times z - x\times z) + (y\times z - y\times z)$	zero and identity
$=$	$x + y$	

We might sometimes want to find an equivalent expression that isn't simpler; to remove the directionality I'll say “calculation” rather than “simplification”. We can use operators other than $=$ down the left side of the calculation; we can even use a mixture of operators, as long as there is transitivity. For example, the calculation

	$x\times(x + 2)$	distribute
$=$	$x^2 + 2\times x$	add and subtract 1
$=$	$x^2 + 2\times x + 1 - 1$	factor
$=$	$(x + 1)^2 - 1$	a square is nonnegative
$\geq$	-1	

tells us

$$x\times(x + 2) \geq -1$$

Boolean calculation is similar. For example,

	$(a \Rightarrow b) \vee (b \Rightarrow a)$	replace implications
$\equiv$	$\neg a \vee b \ \vee \ \neg b \vee a$	$\vee$ is symmetric
$\equiv$	$a \vee \neg a \ \vee \ b \vee \neg b$	excluded middle, twice
$\equiv$	$true \vee true$	$\vee$ is idempotent
$\equiv$	$true$	

And so $(a \Rightarrow b) \vee (b \Rightarrow a)$ has been simplified to *true* , which is to say it has been proven. Here is another example.

	$\exists n \cdot\ n + n^2 = n^3$	instance
$\Leftarrow$	$0 + 0^2 = 0^3$	arithmetic
$\equiv$	*true*	

And so $\exists n \cdot\ n + n^2 = n^3 \Leftarrow \mathit{true}$, and so $\exists n \cdot\ n + n^2 = n^3$ is proven.

Solving simultaneous equations can also be done as a boolean calculation. For example,

	$x + x \times y + y = 5\ \wedge\ x - x \times y + y = 1$	subtract and add $2 \times x \times y$ in first equation
$\equiv$	$x - x \times y + y + 2 \times x \times y = 5\ \wedge\ x - x \times y + y = 1$	use second equation to simplify first
$\equiv$	$1 + 2 \times x \times y = 5\ \wedge\ x - x \times y + y = 1$	
$\equiv$	$2 \times x \times y = 4\ \wedge\ x - x \times y + y = 1$	
$\equiv$	$x \times y = 2\ \wedge\ x - x \times y + y = 1$	use first equation to simplify second
$\equiv$	$x \times y = 2\ \wedge\ x - 2 + y = 1$	
$\equiv$	$x \times y = 2\ \wedge\ x + y = 3$	
$\Leftarrow$	$x{=}1 \wedge y{=}2$	

These examples show that simplifying, proving, and solving are all the same: they are all just calculation.

When an expression is too long to fit on one line, it must be nicely formatted for easy reading, and when a hint is too long to fit on the remainder of a line, it can be written on as many lines as it takes, but we do not consider formatting further here. One point worth mentioning is that subcalculations (if boolean, they are called subproofs or lemmas) can save copying unchanged parts of a calculation through many lines. These subcalculations can be done in another place and referenced, or they can be done in-place, nicely formatted, to provide a structured calculation (structured proof). By far the best way to handle subcalculations is provided by window inference systems [15,2], which open a new window for each subcalculation, keep track of its sense (direction), and make its context available. For example, in solving the simultaneous equations, we used the second equation to simplify the first, and then the first to simplify the second.

In this brief introduction to boolean calculation, we have not taken the time to present all the rules. For a complete presentation, the reader is referred to [9]. A research monograph that uses calculational proof is [4]. A textbook on discrete math that uses calculational proof is [6]. For further discussion of calculational proofs see [12].

Traditional Terminology

Formal logic has developed a traditional terminology that its students are required to learn. There are terms which are said to have values. There are formulas, also known as propositions or sentences, which are said not to have values, but instead to be true or false. Operators (+, –) join terms, while connectives ($\wedge$, $\vee$) join formulas. Some terms are boolean, and they have the value *true* or *false*, but that's different from being true or false. It is difficult to find a definition of predicate, but it seems that a boolean term like $x=y$ stops being a boolean term and mysteriously starts being a predicate when we admit the possibility of using quantifiers ($\exists$, $\forall$). Does $x+y$ stop being a number term if we admit the possibility of using summation and product (Σ, Π)? There are at least three different equal signs: $=$ for terms, and $\Leftrightarrow$ and $\equiv$ for formulas and predicates, with one of them carrying an implicit universal quantification. We can even find a peculiar mixture in some textbooks, such as the following:

$$a+b = a \vee a+b = b$$

Here, a and b are boolean variables, $+$ is a boolean operator (disjunction), $a+b$ is a boolean term (having value *true* or *false*), $a+b = a$ and $a+b = b$ are formulas (so they are true or false), and finally $\vee$ is a logical connective.

Fortunately, in the past few decades there has been a noticeable shift toward erasing the distinction between being true or false and having the value *true* or *false*. It is a shift toward the calculational style of proof. But we have a long way to go yet, as I find whenever I ask my beginning students to prove something of the form $a \oplus b$ where $\oplus$ is pronounced "exclusive or". They cannot even start because they expect something that looks grammatically like a sentence. If I change it to either of the equivalent forms $(a \oplus b) \equiv true$ or $a \not\equiv b$ they are happy because they can read it as a sentence with a verb. But $(a \not\equiv b) \equiv true$ confuses them again because it seems to have too many verbs. If I ask them to prove something of the form $a \vee b$, they take an unwittingly constructivist interpretation, and suppose I want them to prove a or prove b because that is what "do a or b" means in English. The same lack of understanding can be found in many introductory programming texts where boolean expressions are not taught in their generality but as comparisons because comparisons have verbs. We find

while *flag*=*true* **do** *something*

but not the equivalent, simpler, more efficient

while *flag* **do** *something*

because *flag* isn't the right part of speech to follow **while**. Our dependence on natural language for the understanding of boolean expressions is a serious impediment.

Traditional Notations

Arithmetic notations are reasonably standard throughout the world. The expression

738 + 45 = 783

is recognized and understood by schoolchildren almost everywhere. But there are no standard boolean notations. Even the two boolean constants have no standard symbols. Symbols in use include

true	t	tt	T	1	0	1=1
false	f	ff	F	0	1	1=2

Quite often the boolean constants are written as 1 and 0, with + for disjunction, juxtaposition for conjunction, and perhaps – for negation. With this notation, here are some laws.

$x(y+z) = xy + xz$

$x + yz = (x+y)(x + z)$

$x + -x = 1$

$x(-x) = 0$

The overwhelming reaction of algebraists to notational criticisms is: it doesn't matter which symbols are used. Just introduce them, and get on with it. But to apply an algebra, one must recognize the patterns, matching laws to the expression at hand. The laws have to be familiar. The first law above coincides with number algebra, but the next three clash with number algebra. It takes an extra moment to think which algebra I am using as I apply a law. The logician R.L. Goodstein [5] chose to use 0 and 1 the other way around, which slows me down a little more. A big change, like using + as a variable and x as an operator, would slow me down a lot. I think it matters even to algebraists because they too have to recognize patterns. To a larger public, the reuse of arithmetic symbols with different meanings is an insurmountable obstacle. And when we mix arithmetic and boolean operators in one expression, as we often do, it is impossible to disambiguate.

The most common notations for the two boolean constants found in programming languages and in programming textbooks seem to be *true* and *false*. I have two objections to these symbols. The first is that they are English-based and clumsy. Number algebra could never have advanced to its present state if we had to write out words for numbers.

seven three eight + four five = seven eight three

is just too clumsy, and so is

$true \wedge false \vee true \equiv true$

Clumsiness may seem minor, but it can be the difference between success and failure in a calculus.

My second, and more serious, objection is that the words *true* and *false* confuse the algebra with an application. One of the primary applications of boolean algebra is to formalize reasoning, to

determine the truth or falsity of some statements from the truth or falsity of others. In that application, we use one of the boolean values to represent an arbitrary true statement, and the other to represent an arbitrary false statement. So for that application, it seems reasonable to call them *true* and *false*. The algebra arose from that application, and it is so much identified with it that many people cannot separate them; they think the boolean values really are *true* and *false*. But of course boolean expressions are useful for describing anything that comes in two kinds. We apply boolean algebra to circuits in which there are two voltages. We sometimes say that there are 0s and 1s in a computer's memory, or that there are *true*s and *false*s. Of course that's nonsense; there are neither 0s and 1s nor *true*s and *false*s in there; there are low and high voltages. We need symbols that can represent truth values and voltages equally well.

Boolean expressions have other applications, and the notations we choose should be equally appropriate for all of them. Computer programs are written to make computers work in some desired way. Before writing a program, a programmer should know which ways are desirable and which are not. That divides computer behavior into two kinds, and we can use boolean expressions to represent them. A boolean expression used this way is called a specification. We can specify anything, not just computer behavior, using boolean expressions. For example, if you would like to buy a table, then tables are of two kinds: those you find desirable and are willing to buy, and those you find undesirable and are not willing to buy. So you can use a boolean expression as a table specification. Acceptable and unacceptable human behavior is specified by laws, and boolean expressions have been proposed as a better way than legal language for writing laws [1].

For symbols that are independent of the application, I propose the lattice symbols $\top$ and $\bot$, pronounced “top” and “bottom”. Since boolean algebra is the mother of all lattices, I think it is appropriate, not a misuse of those symbols. They can equally well be used for true and false statements, for high and low voltages (power and ground), for satisfactory and unsatisfactory tables, for innocent and guilty behavior, or any other opposites.

We seem to be settling on the symbols $\wedge$ and $\vee$ for conjunction and disjunction, although they are still not universal. They are explained by the use of the words “and” and “or”; even when they are explained by their “truth tables” we remember them by the fact that $x \wedge y$ is $\top$ exactly when both x and y are $\top$, and similarly for $\vee$.

We are less settled on a symbol for implication. Symbols in use include

$$\rightarrow \quad \Rightarrow \quad \therefore \quad \supset$$

The usual explanation says it means “if then”, followed by a discussion about the meaning of “if then”. Apparently, people find it difficult to understand an implication whose antecedent is *false*;

for example, “If my mother had been a man, I'd be the king of France.” [13]. Such an implication is called “counter-factual”. Some people are uneasy with the idea that *false* implies anything, so some researchers in Artificial Intelligence have proposed a new definition of implication. The following truth table shows both the old and new definitions.

a	*b*	old $a \Rightarrow b$	new $a \Rightarrow b$
true	*true*	*true*	*true*
true	*false*	*false*	*false*
false	*true*	*true*	*unknown*
false	*false*	*true*	*unknown*

where *unknown* is a third boolean value. When the antecedent is *false* , the result of the new kind of implication is *unknown* . This is argued to be more intuitive. I believe this proposal betrays a serious misunderstanding of the use of logic. When someone makes a statement, they are saying that the statement is true. Even if the statement is “if *a* then *b* ” and *a* is known to be false, nonetheless we are being told that “if *a* then *b* ” is true. It is the consequent *b* that is unknown. And that is represented perfectly by the old implication: there are two rows in which *a* is *false* and $a \Rightarrow b$ is *true* ; on one of these rows, *b* is *true* , and on the other *b* is *false* .

In case you think that confusion is just for beginners or philosophers, consider the explanation of implication in *Contemporary Logic Design*, 1994 [11]:

> “As an example, let's look at the following logic statement:
>
> IF the garage door is open
>
> AND the car is running
>
> THEN the car can be backed out of the garage
>
> It states that the conditions — the garage is open and the car is running — must be true before the car can be backed out. If either or both are false, then the car cannot be backed out.”

Even a Berkeley electrical engineering professor can't get implication right.

Implication is best presented as an ordering. If we are calling the boolean values “top” and “bottom”, we can say “lower than or equal to” for implication. It is easy, even for primary school students, to accept that $\bot$ is lower than or equal to $\top$, and that $\bot$ is lower than or equal to $\bot$. With this new pronunciation and explanation, three other neglected boolean operators become familiar and usable; they are “higher than or equal to”, “lower than”, and “higher than”. For lack of a name and symbol, the last two operators have been treated like shameful secrets, and shunned. If we are still calling the boolean values “true” and “false”, then we shall have to call implication “falser than or equal to”, or if you prefer, “as false as”. As we get into boolean expressions that use other types, ordering remains a good explanation: $x<4$ is falser than or equal to $x<6$, as a

sampling of evaluations illustrates (try $x = 3, 5, 7$). I have tried using the words "stronger" and "weaker", saying $x<4$ is stronger than $x<6$; but I find that some of my students have an ethical problem that prevents them from accepting that falsity is stronger than truth.

That implication is the boolean ordering, with $\top$ and $\bot$ at the extremes, is not known to all who use boolean algebra. In the specification language Z [18], boolean expressions are used as specifications. Specification A refines specification B if all behavior satisfying A also satisfies B . Although increasing satisfaction is exactly the implication ordering, the designers of Z defined a different, complicated ordering for refinement where $\top$ is not satisfied by all computations, only by terminating computations, and $\bot$ is satisfied by some computations, namely nonterminating computations. When even they can get it wrong, logic is not well understood or used.

Implication has often been defined as a "secondary" operator in terms of the "primary" operators negation and disjunction:

$$(a \Rightarrow b) \equiv \neg a \vee b$$

Proofs about implications proceed by getting rid of them in favor of the more familiar negation and disjunction, as we did earlier in an example. This avoids the philosophical explanation, but it makes an unsupportable distinction between "primary" and "secondary" operators, and hides the fact that it is an ordering. When we learn that implication is an ordering, proofs about implications become shorter and easier.

If we present implication as an ordering, as I prefer, then we face the problem of how to use this ordering in the formalization of natural language reasoning. To what extent does the algebraic operator "lower than or equal to" correspond to the English word "implication"? Philosophers and linguists can help, or indeed dominate in this difficult and important area. But we shouldn't let the complexities of this application of boolean algebra complicate the algebra, any more than we let the complexities of the banking industry complicate the definition of arithmetic.

Symmetry and Duality

In choosing binary infix symbols, there is a simple principle that really helps our ability to calculate: we should choose symmetric symbols for symmetric operators, and asymmetric symbols for asymmetric operators, and choose the reverse of an asymmetric symbol for the reverse operator. The benefit is that we get a lot of laws for free: we can write an expression backwards and get an equivalent expression. For example, $x + y < z$ is equivalent to $z > y + x$. By this principle, the arithmetic symbols $+ \times < > =$ are well chosen but $-$ and $\neq$ are not. The boolean symbols $\wedge \vee \Rightarrow \Leftarrow \equiv \oplus$ are well chosen, but $\not\equiv$ is not.

Duality can be put to use, just like symmetry, if we use vertically symmetric symbols for self-dual operators, and vertically asymmetric operators for non-self-dual operators with the vertical reverse for their duals. The laws we get for free are: to negate an expression, turn it upside down. For example, $(\top \wedge -\bot) \vee \bot$ is the negation of $(\bot \vee -\top) \wedge \top$ if you allow me to use the vertically symmetric symbol $-$ for negation, which is self-dual. There are two points that require attention when using this rule. One is that parentheses may need to be added to maintain the precedence; but if we give dual operators the same precedence, there's no problem. The other point is that variables cannot be flipped, so we negate them instead (since flipping is equivalent to negation). The well-known example is deMorgan's law: to negate $a \vee b$, turn it upside down and negate the variables to get $-a \wedge -b$. By this principle, the symbols $\top$ $\bot$ $-$ $\wedge$ $\vee$ are well chosen, but $\Rightarrow$ $\Leftarrow$ $\equiv$ $\not\equiv$ $\oplus$ are not. By choosing better symbols we can let the symbols do some of the work of calculation, moving it to the level of visual processing.

Booleans and Numbers

I have long thought it was a mistake to identify booleans with numbers, even if just by the reuse of symbols. It's a type error. The C language continues the mistake. In C we can write (1 && 1) + 1 and get 2. I have recently changed my mind. I now think the association —even the identification— between booleans and numbers is right, but not the association we are used to.

I like to prove things about the execution time of programs, and for that purpose I use a number system extended with an infinity (because that's the execution time of some programs). For some purposes I use integers extended with infinity, and for others I use reals extended with infinity. For a list of axioms of this arithmetic, please see [8,9]. Here I mention only that infinity is maximum $x \leq \infty$ and it absorbs additions $\infty+1=\infty$. For my purposes, I have not needed a lot of infinite cardinalities; a single infinity is enough. The association I want to make between booleans and numbers is the following.

	boolean	number	
top	$\top$	∞	infinity
bottom	$\bot$	$-\infty$	minus infinity
negation	$\neg$	$-$	negation
conjunction	$\wedge$	$\downarrow$	minimum
disjunction	$\vee$	$\uparrow$	maximum
“nand”	$\Uparrow$	$\updownarrow$	negation of minimum
“nor”	$\Downarrow$	$\updownarrow$	negation of maximum
implication	$\Rightarrow$	$\leq$	order
reverse implication	$\Leftarrow$	$\geq$	reverse order
strict implication	»	$<$	strict order

strict reverse implication	«	>	strict reverse order
equivalence	≡	=	equality
exclusive or	⊕	≠	inequality

I have temporarily invented a few symbols to fill in some gaps, but my choice of symbols is not the point at the moment. The remaining three unary operators and six binary operators are degenerate, so I have not included them. With this association, all number laws employing only these operators correspond to boolean laws. For example,

boolean law	number law
$\top \equiv \neg\bot$	$\infty = - -\infty$
$a \equiv \neg\neg a$	$x = - -x$
$a \vee \top \equiv \top$	$x \uparrow \infty = \infty$
$a \wedge \bot \equiv \bot$	$x \downarrow -\infty = -\infty$
$a \vee \bot \equiv a$	$x \uparrow -\infty = x$
$a \wedge \top \equiv a$	$x \downarrow \infty = x$
$a \Rightarrow \top$	$x \leq \infty$
$\bot \Rightarrow a$	$-\infty \leq x$
$a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$	$x \uparrow (y \downarrow z) = (x \uparrow y) \downarrow (x \uparrow z)$
$a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$	$x \downarrow (y \uparrow z) = (x \downarrow y) \uparrow (x \downarrow z)$
$a \vee b \equiv \neg(\neg a \wedge \neg b)$	$x \uparrow y = -(-x \downarrow -y)$
$a \wedge b \equiv \neg(\neg a \vee \neg b)$	$x \downarrow y = -(-x \uparrow -y)$

There are, however, boolean laws that do not correspond to number laws. For example,

boolean law	number non-law
$\top$	∞
$\neg\bot$	$- -\infty$
$a \vee \neg a \equiv \top$	$x \uparrow -x = \infty$
$a \wedge \neg a \equiv \bot$	$x \downarrow -x = -\infty$
$(a \equiv \top) \equiv a$	$(x = \infty) = x$

Number algebra has developed by the desire to solve equations, or more generally, to solve boolean expressions. This has resulted in an increasing sequence of domains, from naturals to integers to rationals to reals to complex numbers. As we gain solutions, we lose laws.

small domain	⟷	large domain
more laws	⟷	fewer laws
fewer solutions	⟷	more solutions

This is because a law is essentially a universal quantification, and a boolean expression to be solved is essentially an existential quantification.

law: $\forall$ *variables*: *domains*· *boolean expression*

solution: $\exists$ *variables*: *domains*· *boolean expression*

As the domain of an operation or function grows, we do not change its symbol; addition is still denoted + as we go from naturals to complex numbers. I will not argue whether the naturals are a subset of the complex numbers or just isomorphic to a subset; for me the question has no meaning. But I do argue that it is important to use the same notation for natural 1 and complex 1 because they behave the same way, and for natural + and complex + because they behave the same way on their common domain. To be more precise, all laws of complex arithmetic that can be interpreted over the naturals are laws of natural arithmetic, and all equations (or more generally, boolean expressions) over the naturals retain the same solutions over the complex numbers. The reason we must use the same symbols is so that we do not have to relearn all the laws and solutions as we enlarge or shrink the domain.

I have been hammering on a point that I expect is not contentious. If I have your agreement, then you must conclude, as I must, that the symbols of boolean algebra and arithmetic must be unified. The question whether boolean is a different type from number is no more relevant than the question whether natural and integer are different types. What's important is that laws and solutions are learned once, in a unified system, not twice in conflicting systems. And that matters both to professional mathematicians who must apply laws and solve, and to primary school students who must struggle to learn what will be useful to them.

Unified Algebra

Here is my proposal for the symbols of a unified algebra.

	unified	
top	$\top$	infinity
bottom	$\bot$	minus infinity
negation	$-$	negation
conjunction	$\wedge$	minimum
disjunction	$\vee$	maximum
“nand”	$\triangle$	negation of minimum
“nor”	$\triangledown$	negation of maximum
implication	$\leq$	order
reverse implication	$\geq$	reverse order
strict implication	$<$	strict order
strict reverse implication	$>$	strict reverse order
equivalence	$=$	equality
exclusive or	$\neq$	inequality

The symbols $-\ \leq\ \geq\ <\ >\ =$ are world-wide standards, used by school children in all countries, so I dare not suggest any change to them. The symbol $\neq$ for inequality is the next best known, but I have dared to stand up the slash so that all symmetric operators have symmetric symbols and all asymmetric operators have asymmetric symbols. (Although it was not a consideration, $\not=$ also looks more like $\oplus$.) There are no standard symbols for minimum and maximum, so I have used the boolean conjunction and disjunction symbols. The "nand" symbol is a combination of the "not" and "and" symbols, and similarly for "nor". Duality has been sacrificed to standards; the pair $\leq\ <$ are duals, so they ought to be vertical reflections of each other; similarly the pair $\geq\ >$, and also $=\ \not=$. Since we now have a unified boolean and number algebra, I might mention that addition and subtraction are self-dual, and happily $+$ and $-$ are vertically symmetric; multiplication is not self-dual, but $\times$ is unfortunately vertically symmetric.

Having unified the symbols, I suppose we should also unify the terminology. I vote for the number terminology in the right column, except that I prefer to call $\top$ and $\bot$ "top" and "bottom".

In the unified algebra, the fact that $x=-x$ has no boolean solution but does have an integer solution is no more bothersome than that $x^2=2$ has no integer solution but does have a real solution. The fact that $x\not=-x$ is a boolean law but not an integer law is no more bothersome than that $x^2\not=2$ is an integer law but not a real law.

As an example, the arithmetic expression $x-y$ varies directly with x and inversely with y . Thus if we increase x , we increase $x-y$, and if we decrease y we increase $x-y$. So we can calculate as follows:

	$x-y$	increase x to $x+1$ and so increase the whole expression
$\leq$	$(x+1)-y$	decrease y to $y-1$ and so increase the whole expression
$\leq$	$(x+1)-(y-1)$	

Similarly the boolean expression $x\geq y$ varies directly with x and inversely with y . So we can calculate as follows:

	$x\geq y$	increase x to $x+1$ and so increase the whole expression
$\leq$	$(x+1)\geq y$	decrease y to $y-1$ and so increase the whole expression
$\leq$	$(x+1)\geq(y-1)$	

It is exactly the same calculation. By unifying number algebra with boolean algebra we carry our ability to calculate over from numbers to booleans.

from Informal to Formal

Many mathematical notations began their life as abbreviations for some words. For example, = was introduced by Robert Recorde in [2] to mean "is equal to":

> "And to avoide the tediouse repetition of these woordes "is equalle to" I will lette as I doe often in woorke bse, a paire of paralleles or Gemowe [twin] lines of one lengthe, thus: =, because noe 2 thynges, can be moare equalle."

Later, = became associated with some algebraic properties, namely reflexivity, symmetry, transitivity, and substitutivity. Today, it is defined by those properties, not as an abbreviation for some words. Someone might say that Alan and Bob are equal tennis players because they have played each other 100 times, and each has won 50 matches. They might similarly say that Bob and Carl are equal tennis players because they too have played each other 100 times, and each has won 50 matches. But this kind of equality is not transitive. As it happens, Alan and Carl are unequal tennis players: they have played each other 100 times, and Alan has won 75 matches. Because of the lack of transitivity, no mathematician today would use = for tennis equality.

In the notation commonly used for small sets, such as $\{1, 3, 7\}$, the comma was introduced as just punctuation, not as a mathematical operator. As soon as the notation is introduced, we must say that the order in which elements are written is irrelevant so that $\{1, 2\} = \{2, 1\}$; the way to say that formally is $A,B = B,A$ (comma is symmetric or commutative). We must also say that repetitions of elements are irrelevant so that $\{3, 3\} = \{3\}$; the way to say that formally is $A,A = A$ (comma is idempotent). And we should say that comma is associative $A,(B,C) = (A,B),C$ so that parentheses are unnecessary. Evidently the comma can be seen as a mathematical operator with algebraic properties that aggregates elements into a structure that is simpler, more primitive, than sets; let us call them bunches. Even the curly braces can be seen as an operator that applies to bunches and makes a set; its inverse ~ applies to a set and makes a bunch: $\sim\{A\} = A$.

When a child first learns about sets, there is often an initial hurdle: that a set with one element is not the same as the element. It would be easier to present a set as packaging: a package with an apple in it is obviously not the same as the apple. Just as $\{2\}$ and 2 differ, so $\{2,7\}$ and 2,7 differ. Bunch theory tells us about aggregation; set theory tells us about packaging. The two are independent.

Apart from being cute, are bunches useful? The subject of functional programming has suffered from an inability to express nondeterminism conveniently. To say something about a value, but not pin it down completely, one can express the set of possible values. Unfortunately, sets do not reduce properly to the deterministic case; in this context it is again a problem that a set containing one element is not equal to the element. What is wanted is bunches. One can always regard a

bunch as a “nondeterministic value”. Bunches can also be used as a “type theory” with the advantage that it is unnecessary to duplicate the operators of the value space at the type level because the two are unified. And finally, the easiest way to present sets is via bunches. For details see [8,9]. Formalization of the lowly comma leads to a beautiful and useful algebra.

We have just seen two examples of formalization, one from the past and one from the future. Now here's an example of a formalization gone astray: functions defined as sets of ordered pairs. This way of defining functions is part of the very interesting demonstration that all of mathematics can be based on sets. The demonstration requires us to make a set-model of functions, and numbers, and everything else. For example, the natural numbers can be equated to sets as follows:

$0 = \varnothing$ (the empty set)

$n{+}1 = n \cup \{n\}$

So, for example, $3 = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$. Few people would say that 3 really is the set $\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$; the set-model of natural numbers was constructed just to serve this one demonstration. Set union and set intersection of two numbers in this model are not useful operations. Numbers are best formalized, not by building a set-model, but by an algebra showing how they participate in arithmetic operations. Set union and intersection are no more useful for combining functions than numbers; much more useful are the laws of application and function composition. But the set-model of functions has somehow taken root in the current mathematical culture; many people (and textbooks) say that a function really is a set of ordered pairs. A useful formalization is not one that answers the question “what is it?”, but one that answers the question “how do we use it?”.

I write a function, or local scope, according to the following example:

$\langle n\colon nat \rightarrow n{+}1 \rangle$

This is essentially a “lambda-expression”[3], although Church did not use angle brackets and arrows. He borrowed a “hat” notation from Whitehead and Russell, but moved the hat down in front; the most similar available character in the typesetter's tray was λ; thus the lambda calculus was born [16]. Following van de Snepscheut [17], I use angle brackets to delimit scope of the variable. I use an arrow to facilitate the unification of functions with function spaces, which I do not discuss in this paper (see [9]). Next, I want to get rid of the idea that all possible variables (infinitely many of them) already “exist”, and that the function notation “binds” a variable, and any variable that is not bound remains “free”. I prefer the programmer's terminology of “local” and “nonlocal” variables. Variables do not automatically “exist”; they are introduced (rather than bound) with a limited scope by the function notation.

Two notations that have not yet made the transition from informal beginning to formal, calculational tool are the quantifiers $\forall$ and $\exists$. For most mathematicians today they remain abbreviations for the words "for all" and "there exists", and their meaning is just whatever can be understood from those words. The word "all" sounds clear and unambiguous, but there is debate as to whether so-called "undefined" range elements, or other "nonstandard" elements, are included. Existence is even more contentious, as can be seen from the debate between classical and constructive mathematicians. Only a formal definition, equivalent to an automated theorem prover, is clear and unambiguous. Only a formal definition gives us calculation.

Quantifiers

There are several mathematical notations that introduce a local (or bound or dummy) variable. For example,

$$\sum_{x=0}^{\infty} fx \qquad \int_a^b fx\ dx \qquad \forall x\colon D\cdot Px \qquad \{fx \mid x\in D\}$$

The introduction of the local variable and its domain are exactly the job of the function notation, so all expressions requiring a local variable can be uniformly expressed as an operator applied to a function. If the body of a function is a number expression, then we can apply $+$ to obtain the sum of the function results. For example,

$$+\langle n\colon nat \to 1/2^n\rangle$$

There is no syntactic ambiguity caused by this use of $+$, so no need to employ another symbol Σ for addition. We can apply any binary associative symmetric operator, such as

$$\times\langle n\colon nat \to 1/2^n\rangle$$
$$\wedge\langle n\colon nat \to n>5\rangle$$
$$\vee\langle n\colon nat \to n>5\rangle$$

The minimum operator $\wedge$ replaces "for all", and the maximum operator $\vee$ replaces "there exists". By applying $=$ and $\neq$ to functions we obtain the two independent parity operators. Set comprehension and integrals can be treated this same way.

The sum of two rationals is rational; the sum of infinitely many rationals may not be rational. Nonetheless, we continue to use the word "sum" and symbol $+$ in the infinite case. Similarly, I see no need to switch teminology from "minimum" and "maximum" to"greatest lower bound" and "least upper bound" as we generalize $\wedge$ and $\vee$ from two operands to infinitely many; we just have to learn that the minimum and maximum of a set may not be in the set. For example,

$$\wedge\langle n\colon nat \to 1/n\rangle = 0$$
$$\vee\langle n\colon nat \to n\rangle = \top$$

If function f has domain D, then $f = \langle x: D \to fx \rangle$, so quantifications traditionally written

$$\sum_{x:\, D} fx \qquad\qquad \forall x: D \cdot Px$$

which we can now write as

$$+\langle x: D \to fx \rangle \qquad\qquad \wedge\langle x: D \to Px \rangle$$

can be written even more succinctly as

$$+f \qquad\qquad \wedge P$$

Using juxtaposition for composition, deMorgan's laws

$$\neg(\forall x: D \cdot Px) \equiv (\exists x: D \cdot \neg Px) \qquad\qquad \neg(\exists x: D \cdot Px) \equiv (\forall x: D \cdot \neg Px)$$

become

$$-\wedge P = \vee -P \qquad\qquad -\vee P = \wedge -P$$

or even more succinctly

$$(-\wedge) = (\vee -) \qquad\qquad (-\vee) = (\wedge -)$$

The Specialization and Generalization laws say that if $y: D$,

$$(\forall x: D \cdot Px) \Rightarrow Py \qquad\qquad Py \Rightarrow (\exists x: D \cdot Px)$$

They now become

$$\wedge P \leq Py \qquad\qquad Py \leq \vee P$$

which say that the minimum item is less than or equal to any item, and any item is less than or equal to the maximum item. These laws hold for all numbers, not just for the booleans.

Given function f, some function value fx is at least y if and only if the maximum function value fx is at least y. Traditionally, that's an existential quantification equated to a maximum. In unified algebra, it is just factoring. Leaving the domain of f implicit, we write

$$\begin{aligned} & \vee\langle x \to fx \geq y \rangle && \text{factor out } \geq y \\ = \quad & \vee\langle x \to fx \rangle \geq y \\ = \quad & \vee f \geq y \end{aligned}$$

If we go in the other direction, "unfactoring" is called "distribution". And it works whether fx and y are numbers and $\geq$ is the number ordering, or fx and y are booleans and $\geq$ is reverse implication. It's no different from the factoring/distribution law that says the maximum value of $(fx - y)$ equals (the maximum value of fx) $- y$.

$$\begin{aligned} & \vee\langle x \to fx - y \rangle && \text{factor out } -y \\ = \quad & \vee\langle x \to fx \rangle - y \\ = \quad & \vee f - y \end{aligned}$$

If we factor from the other side of the $-$ sign, we have to change maximum to minimum:

$$\begin{aligned} & \vee\langle x \to y - fx \rangle && \text{factor out } y - \\ = \quad & y - \wedge\langle x \to fx \rangle \\ = \quad & y - \wedge f \end{aligned}$$

And similarly

$$\begin{array}{lll} & \vee\langle x \to y \geq fx\rangle & \text{factor out } y \geq \\ = & y \geq \wedge\langle x \to fx\rangle & \\ = & y \geq \wedge f & \end{array}$$

Once again, it works for numbers and booleans equally well. Unified algebra gives us many other factoring/distribution laws just like these (see [9]).

Quantifier Examples

Here's an interesting experiment: ask a colleague if $(\forall x{\cdot}Px) \Rightarrow (\exists y{\cdot}\ Qy)$ is equivalent to $\exists x{\cdot}\ \exists y{\cdot}\ (Px \Rightarrow Qy)$ and then listen to their efforts to find the answer. They probably don't find it obvious. Those who reason informally say things like "suppose all x have property P", and "suppose some y has property Q". They are led into case analyses by treating $\forall$ and $\exists$ as abbreviations for "for all" and "there exists" (as they originally were). Of the very few who reason formally, most don't know many laws; perhaps they start by getting rid of the implications in favor of negation and disjunction, then use deMorgan's laws. Let me rewrite the question in the new notations.

$$(\wedge P \leq \vee Q) \ = \ \vee\langle x \to \vee\langle y \to Px \leq Qy\rangle\rangle$$

We might read the left side as saying that the minimum P is at most the maximum Q, and we might read the right side as saying that some P is at most some Q. Informal readings can be misleading, and we should never attach our understanding to an informal reading, but sometimes we can get inspiration from it. In this case, the reading sounds reasonable enough to suggest we might prove it, and not just for booleans, but for all numbers. Here's the proof:

$$\begin{array}{lll} & \vee\langle x \to \vee\langle y \to Px \leq Qy\rangle\rangle & \text{factor out } Px \leq \\ = & \vee\langle x \to Px \leq \vee Q\rangle & \text{factor out } \leq \vee Q \\ = & \wedge P \leq \vee Q & \end{array}$$

Let L be a list (a function whose domain is an initial segment of the naturals). $+L$ is its sum, and $\vee L$ is its maximum; let $\#L$ be its length. We can say that the average item in the list is less than or equal to the maximum item as follows.

$$\begin{array}{lll} & +L/\#L \ \leq \ \vee L & \text{now apply } >1 \text{ to both sides of the inequality} \\ \leq & (+L/\#L > 1) \leq (\vee L > 1) & \text{multiply by } \#L\text{; distribute } >1 \\ = & (+L > \#L) \leq \vee\langle i \to Li{>}1\rangle & \end{array}$$

leaving the domain implicit. The bottom line is the "pigeon-hole principle"; it says that if the total number of things is greater than the number of places to put them, then some place has more than one thing in it. Notice what has happened: we read $\vee$ as "maximum" on the top line, and as "some" on the bottom line; we read $\leq$ as "less than or equal to" on the top line, and as "if then" on the bottom line.

Here is a further illustration of the benefits of unified algebra. Let f be a function from the naturals to the reals. If f is nondecreasing, then $f0$ is its minimum. Traditionally, this might be written (leaving the domain implicit) as

$$(\forall n\cdot fn \le f(n{+}1)) \;\Rightarrow\; (f0 = MIN\,\{fn \mid 0{\le}n{<}\infty\})$$

Rewriting this in the new notation, and weakening it to say that $f0$ is less than or equal to the minimum, we get

$$\wedge\langle n \to fn \le f(n{+}1)\rangle \;\le\; (f0 \le \wedge f)$$

Now we apply the portation law, which says that for boolean a and any b and c,

$$(a \le (b \le c)) \;=\; (a{\wedge}b \le c)$$

to obtain

$$f0 \wedge \wedge\langle n \to fn \le f(n{+}1)\rangle \;\le\; \wedge f$$

If f happens to have a boolean range, this is induction, more traditionally written

$$f0 \wedge (\forall n\cdot fn \Rightarrow f(n{+}1)) \;\Rightarrow\; (\forall n\cdot fn)$$

Thus we see induction as a special case of a more general law saying that the first item in a nondecreasing sequence is its minimum.

Metalogic

In the study of logic, at or near the beginning, logicians present the very important symbol $\vdash$ to represent theoremhood. I ask you to put yourself in the place of a beginning student. This symbol is applied to a boolean expression just like the boolean operators; but we know all the boolean operators and this isn't one of them. It sometimes has a left operand as well as a right operand, and then the explanation makes it seem just like implication. To say that it is a “meta-operator” just labels it, and doesn't explain it. Saying that it applies to the form, rather than the meaning, is confusing too, since the entire point of the algebra is to enable us to work with the form and ignore the meaning. The distinction between meta-notations and the object notations is not easily appreciated.

Proof rules are sometimes presented using a horizontal bar, with hypotheses above and conclusion below. Is there any difference between

$$\frac{A,\ B}{C} \qquad \text{and} \qquad A,\ B \vdash C$$

One sees these two forms used together in one rule, for example, the Deduction Theorem

$$\frac{A \vdash B}{\vdash A \Rightarrow B}$$

saying that $\vdash$ really is the same as implication at the meta-level. But so is the horizontal bar.

To apply an operator to the form of an expression, we do not need any new kind of operator. Rather, we need to do exactly what Gödel did when he encoded expressions, but we can use a better encoding. We need to do exactly what programmers do: distinguish program from data. One person's program may be a compiler writer's data, but when it is data, it is a character string. The character string "$a \vee -a$" can be used as a code for the expression $a \vee -a$. We apply $\vdash$ to character strings so that $\vdash s$ is a theorem when the boolean expression represented by string s is a theorem.

We have a name, "theorem", for a boolean expression that can be simplified to $\top$, and an operator, $\vdash$, whose purpose is to identify theorems. Strangely, logicians have not introduced a name, say "antitheorem", for a boolean expression that can be simplified to $\bot$, and no operator such as $\dashv$, whose purpose is to identify antitheorems. Perhaps its lack is explained by noting that "antitheorem" just means "negation of a theorem" in those logics having negation and an appropriate proof rule. But we bother to name both booleans, even though one is just the negation of the other.

I propose that logicians can improve metalogic in another way, by taking another lesson from programming. Instead of $\vdash$ and $\dashv$, we need only one operator to serve both purposes. It is called an interpreter. I want $\mathbb{I}\, s$ to be a theorem if and only if s represents a theorem, and an antitheorem if and only if s represents an antitheorem. It is related to $\vdash$ and $\dashv$ by the two implications

$$\vdash s \;\le\; \mathbb{I}\, s \;\le\; -\dashv s$$

In fact, if we have defined $\vdash$ and $\dashv$, those implications define $\mathbb{I}$. But I want $\mathbb{I}$ to replace $\vdash$ and $\dashv$ so I shall instead define it by showing how it applies to every form of boolean expression. Here is the beginning of its definition.

$$\mathbb{I}\text{“}\top\text{”} = \top$$
$$\mathbb{I}\text{“}\bot\text{”} = \bot$$
$$\mathbb{I}(\text{“}-\text{”}\ s) = -\,\mathbb{I}\, s$$
$$\mathbb{I}(s\ \text{“}\wedge\text{”}\ t) = \mathbb{I}\, s \wedge \mathbb{I}\, t$$
$$\mathbb{I}(s\ \text{“}\vee\text{”}\ t) = \mathbb{I}\, s \vee \mathbb{I}\, t$$

And so on. In a vague sense $\mathbb{I}$ acts as the inverse of quotation marks; it "unquotes" its operand. That is what an interpreter does: it turns passive data into active program. It is a familiar fact to programmers that we can write an interpreter for a language in that same language, and that is just what we are doing here. Interpreting (unquoting) is exactly what logicians call Tarskian semantics. In summary, an interpreter is a better version of $\vdash$, and strings make meta-level operators unnecessary.

Using $\mathtt{I}$, the famous Gödel incompleteness proof is just 3 lines. Suppose that every boolean expression is either a theorem or an antitheorem (a complete logic), and define Q by

$$Q = \text{“}-\mathtt{I}Q\text{”}$$

Then

$$\begin{aligned} & \mathtt{I}Q && \text{replace } Q \text{ with its equal} \\ = \quad & \mathtt{I}\text{“}-\mathtt{I}Q\text{”} && \mathtt{I} \text{ unquotes} \\ = \quad & -\mathtt{I}Q \end{aligned}$$

which proves a boolean expression equal to its negation, showing the logic to be inconsistent. A consistent logic in which we can define an interpreter, and in which we can replace an expression with its equal, must be incomplete. For further details of this version of Gödel's incompleteness theorem, see [7].

You cannot learn a programming language by reading an interpreter for it written in that same language. And you cannot learn logic, or a logic, by reading an interpreter for it written in logic. Not only is it inscrutable to a novice, but also it may be subject to more than one interpretation. We don't present number algebra with the aid of a meta-operator that applies to number expressions and results in their values, and we should not present logic that way. We don't need the word “theorem”, which means “boolean expression that can be simplified to $\mathtt{T}$”, any more than we need a word that means “number expression that can be simplified to 19”. I think boolean algebra should be presented as number algebra is presented, with a little natural language and a lot of laws, because laws don't use any meta-notations.

Terms of Honor

My final comment concerns mathematical terminology intended to honor mathematicians. In some parts of mathematics it is standard: Lie algebra, Stone algebra, Cartesioan product, Jordan decomposition, Cayley transform, Hilbert space, Banach space, Hausdorff space, Borel measure, Lebesgue integration, Fredholm index, and so on. It is well known that the person so honored is sometimes the wrong person; often it is only one of many who equally deserve to have their names attached to the idea. I suspect that sometimes the intention is not so much to honor a person as to use the person's prestige to lend respectability to an idea. Even when the intention is to honor, the effect is to obscure and make the mathematics forbidding and inaccessible. It may be argued that this is good, keeping the uninitiated from thinking they understand when they don't, but I reject that argument as elitist. I know what nand and nor are, but I forget which is the Scheffer stroke and which the Pierce arrow. To say that an operator is symmetric or commutative is much more descriptive and understandable than calling it Abelian. DeMorgan's laws would be better named duality laws. We who are used to the terms forget what a barrier they pose to beginners.

The term “boolean algebra” honors George Boole. It is popularly thought that the word “algebra” honors someone, but it comes from an arabic word meaning “the reintegration and reunion of broken parts”. In any case, the word is now standard, known by people everywhere. The best way to honor George Boole is to make the algebra that he created a well known and well used tool, and to do that we might have to remove his name from it, and give it a more descriptive and accessible name, like “binary algebra”.

Conclusions

Logic has been well studied and is now well understood, but it is not well used. Programmers learn that logic is a foundation of programming, but they don't often use it to program. Mathematicians study about logic, but they don't often use it in their proofs. Logic is a tool, like a knife. People have looked at it from every angle; they've described how it works at great length; now it's time to pick it up and use it. To use logic well, one must learn it early, and practice a lot. Fancy versions of logic, such as three-valued logic, temporal logic, and metalogic, can be left to university study, but there is a simple basic algebra that can be taught early and used widely.

Number algebra is used by scientists and engineers everywhere. It is used by economists and architects. It is taught first to 6-year olds, without a meta-notation, very concretely as addition and subtraction of numbers. Then variables and equations are introduced, and always the applications are emphasized. As a result of that early and long education, scientists and engineers and mathematicians are comfortable with it. Boolean algebra, or logic, can be equally useful if it is taught the same way. At present, it is not in a good state for presentation to a wide audience. We need to simplify the terminology, get rid of the meta-notations, adopt the view that proof is calculation, choose some good symbols, detach it from its dominant application in which the boolean values represent true and false statements, free it from philosophy, and explain it as algebra.

There is a small advantage to choosing uniquely boolean symbols: we can give them a precedence after the arithmetic operators, which reduces the need for parentheses. On the other hand, there is a large advantage to uniting boolean and number symbols in the way I have suggested: the laws and solutions are familiar and can be interpreted either as booleans or numbers. In addition, by placing booleans in the same context as numbers, we move quickly away from philosophical explanations, and we are less likely to introduce strange kinds of implication or strange kinds of logic. The fact that the booleans can be embedded in the extended integers just as smoothly as the integers are embedded in the rationals seems a compelling reason to do so.

Quantifiers can be simplified, made uniform, and generalized by treating them as operators on functions. We should stop speaking about "existence", and speak instead about the maximum of a function. Similarly, we should stop speaking about "all", and speak instead about the minimum of a function. We should stop trying to say what functions and other mathematical ideas are, and say instead how to write them and use them.

An interpreter serves the same purpose as the meta-level theoremhood operator with the added advantage that it gives antitheoremhood as well as theoremhood. And by applying it to strings, we don't need to introduce a separate meta-level of operators. Metalogic is an advanced topic, not a good introduction to logic for those who are new to the subject.

Acknowledgment

Theo Norvell provided references [11] and [13].

References

[0] Unfortunately, 500 year old algebra texts are hard to find. This is not a quotation, but my own creation. I think it is representative of the work of the time.

[1] L.E.Allen: "Symbolic Logic: a Razor-Edged Tool for Drafting and Interpreting Legal Documents", *the Yale Law Journal* v.66 p.833-879, 1957

[2] R.Back, J.Grundy, J.vonWright: "Structured Calculational Proof", to appear in *Formal Aspects of Computing*, Springer

[3] A.Church: "the Calculi of Lambda-Conversion", *Annals of Mathematical Studies* v.6, Princeton University Press 1941

[4] E.W.Dijkstra, C.S.Scholten: *Predicate Calculus and Program Semantics*, Springer-Verlag 1990

[5] R.L.Goodstein: *Development of Mathematical Logic*, Springer-Verlag 1971

[6] D.Gries, F.B.Schneider: *a Logical Approach to Discrete Math*, Springer-Verlag, 1993

[7] E.C.R.Hehner: "Beautifying Gödel", chapter 18 in *Beauty is our Business, a birthday tribute to Edsger Dijkstra*, Springer-Verlag, 1990

[8] E.C.R.Hehner: *a Practical Theory of Programming*, Springer-Verlag 1993

[9] E.C.R.Hehner: "Unified Algebra", to appear

[10] I.N.Herstein: *Topics in Algebra* p.323, Blaisdell 1964

[11] R.H.Katz: *Contemporary Logic Design*, Benjamin Cummings 1994 p.10

[12] L.Lamport: "How to Write a Proof", *Amer. Math. Monthly* 102(7) 600-608 1995 Aug.

[13] C.Navarre: quoted in Barbara W. Tuchman: *a Distant Mirrror: the Calamitous Fourteenth Century*, Knopf 1978

[14] R.Recorde: *the Whetstone of Witte*, London 1557, reprinted by Da Capo Press Amsterdam 1969

[15] P.J.Robinson, J.Staples: "Formalizing a Hierarchical Structure of Practical Mathematical Reasoning", *J. Logic Comput.* 3(1):47-61, February 1993

[16] J.B.Rosser: "Highlights of the History of the Lambda-Calculus", *Annals of the History of Computing* v.6 n.4 p.337-349, 1984 October

[17] J.L.A.van de Snepscheut: *What Computing is All About*, Springer-Verlag, 1993

[18] J.M.Spivey: *the Z Notation: a Reference Manual*, Prentice-Hall, 1989