E-ISSN: 2706-8927
P-ISSN: 2706-8919
IJAAS 2019; 1(1): 128-132
Received: 26-05-2019
Accepted: 27-06-2019

Dr. Daya Shankar Pratap
Research Scholar, Department of Mathematics, JP University, Chapra, Bihar, India

Corresponding Author:
Dr. Daya Shankar Pratap
Research Scholar, Department of Mathematics, JP University, Chapra, Bihar, India

Study of Unified algebra

Dr. Daya Shankar Pratap

Abstract
I present a unified algebra that includes what are commonly called boolean algebra, number algebra, sets, lists, functions, quantification, type theory, and limits; this mathematics forms the foundation for much of computer science. I present the notations and the rules for the conduct of algebra, but it is not the purpose here to explore the possibilities for their use. I am laying foundations, not building upon them; I am designing the instrument, not playing the music. To appreciate the algebra, I rely on the reader's experience in using algebra.

Keywords: Unified algebra

Introduction
The algebra is presented from the very beginning, leaving out nothing. That makes the early parts of the presentation very basic, but readers may appreciate the care and effort required to design a simple and general algebra. And it's a nontrivial problem to get the presentation started without ever saying “trust me for now, I'll make this clear later”. Anyone interested in implementation on a computer must pay attention to micro-mechanical detail. The viewpoint I adopt throughout is formalist, as required for implementation.
I begin with boolean algebra, renamed “binary algebra”, and its two extremes, renamed “top” and “bottom”. That's the only new terminology. By contrast, standard terminology that I won't be using includes: boolean, true, false, proposition, sentence, term, formula, conjunction, conjunct, disjunction, disjunct, implication, implies, antecedent, consequent, axiom, theorem, lemma, proof, inference, entailment, syntax, semantics, valid, predicate, quantifier, universal, existential, and existence. I consider symbols and terminology to be a cost, not a benefit, when defining mathematical structures. Unified algebra gives us much more mathematics for less cost than usual.

Algebra
I will soon introduce binary algebra, ternary algebra, number algebra, the algebra of some data structures, and function algebra. In this section I say what is common to all of them.

Expressions and Values
An algebra consists of expressions, which are used to express values in the application domain. For example, the values may be amounts of water, or voltage, or frequency of vibration, or guilt and innocence. Here are four definitions that precede all choice of symbols and rules of any algebra.

- Consistency: at most one value can be determined for each expression
- Completeness: at least one value can be determined for each expression
- Expressiveness: at least one expression can be determined for each value
- Uniqueness: at most one expression can be determined for each value

We must never use an expression to express more than one value; to do so would be a serious error called inconsistency. Sometimes we may not say what value an expression expresses; that is called incompleteness. For example, we will not be able to determine the value of 0/0. (I prefer to avoid the question of whether 0/0 has no value, or has a value but we cannot say what it is.) Consistency is essential; completeness is not. Expressiveness is desirable; uniqueness is not. In general, several expressions may represent the same value. When we say that 2+3 is 5, we do not mean that 2+3 and 5 are the same expression; clearly they are not. We mean that the value represented by 2+3 is the value represented by 5. When we say that 2+3 has value 5, we again mean that expression 2+3 represents the same value

that expression 5 represents. We might just as well say that 5 has value 2+3.

Expression structure

An expression can be a part of a larger expression, in which case it is called a "subexpression" of the larger expression. One way to make a larger expression from a subexpression is to write a symbol, such as –, followed by the subexpression. The symbol is called an "operator", and the subexpression is called its "operand". Another way to make a larger expression is to write two subexpressions with a symbol, such as + between them.

Placing operators between operands makes the structure of some expressions ambiguous. For example, 2+3×4 might mean that 2 and 3 are added, and then the result is multiplied by 4, or that 2 is added to the result of multiplying 3 by 4. To say which is meant, we can use parentheses: either (2+3)×4 or 2+(3×4). To prevent a clutter of parentheses, we decide on an order of evaluation. Here is the order of evaluation of all operators in this paper.

0	constants a ⊥ 0 1 3.14 and so on variables *x y* and so on bracketed expressions ( ) { } [ ] ⟨ ⟩ within which the order of evaluation again applies	
1	juxtaposition *f x*	left to right
2	one operand – ¢ $ ~ ¼ # ⟶ ∧ ∨ = + § + × ◊	right to left
	two operands ⟶	right to left
	subscript x_n superscript x^n	right to left
3	two operands × / ∧ ∨ ¦–	left to right
4	two operands + – +	left to right
5	two operands ,,.. ‘ ; ;.. | three operands	
6	two operands = + < > ≤ ≥ : ∈	

In the order of evaluation, two-operand + can be found on level 4, and two-operand × on level 3; that means, in the absence of parentheses, evaluate two-operand × before two-operand +. The example 2+3×4 therefore means the same as 2+(3×4). Within levels 1, 3, and 4 evaluation is from left to right. Within level 2 evaluation is from right to left. On level 6, $x = y = z$ means the same as $(x = y)∧(y = z)$, and similarly for the other operators and mixtures of operators on that level.

Binary algebra

The expressions of binary algebra are called "binary expressions". Binary expressions can be used to represent anything that comes in two kinds, such as true and false statements, high and low voltage, satisfactory and unsatisfactory computations, innocent and guilty behavior, north and south poles of magnets. In any application of binary algebra, the two things being represented are called the "binary values". For example, in one application the binary values are truth and falsity; in another they are innocence and guilt. Binary expressions include:

a	"top"
⊥	"bottom"
–x	"negate x"
x = y	"x equal y"
x + y	"x differ y"
x<*y*	"*x* below *y*"
x>*y*	"*x* above *y*"
x≤*y*	"*x* at most"
x≥*y*	"*x* at least *y*"
x∧*y*	"*x* min *y*" (memory aid: the symbol doesn't hold water)
x∨*y*	"*x* max *y*" (memory aid: the symbol does hold water)
x y	"*x* neg min"
x y	"*x* neg max *y*"
x<*y*>*z*	"*x* if *y* else *z*"

The two simplest binary expressions are a and ⊥. Expression a represents one binary value, and expression ⊥ represents the other. In the other binary expressions, the variables *x*, *y*, and *z* may be replaced by any binary expressions. Whichever value is represented by expression *x*, expression –*x* represents the other value. This rule can be shown with the aid of a value table.

x	⊤	⊥
–*x*	⊥	⊤

This table says that –a represents the same value that ⊥ represents, and that –⊥ represents the same value that a represents. We can similarly show how to evaluate other binary expressions.

xy	⊤⊤	⊤⊥	⊥⊤	⊥⊥
x=*y*	⊤	⊥	⊥	⊤
x≠*y*	⊥	⊤	⊤	⊥
x<*y*	⊥	⊥	⊤	⊥
x>*y*	⊥	⊤	⊥	⊥
x≤*y*	⊤	⊥	⊤	⊤
x≥*y*	⊤	⊤	⊥	⊤
x∧*y*	⊤	⊥	⊥	⊥
x∨*y*	⊤	⊤	⊤	⊥
x△*y*	⊥	⊤	⊤	⊤
x▽*y*	⊥	⊥	⊥	⊤

xyz	⊤⊤⊤	⊤⊤⊥	⊤⊥⊤	⊤⊥⊥	⊥⊤⊤	⊥⊤⊥	⊥⊥⊤	⊥⊥⊥
x◁*y*▷*z*	⊤	⊤	⊤	⊥	⊥	⊥	⊤	⊥

Preference

We have two binary values, and so far we have not shown any preference for one over the other. Now we shall show a preference for expressions with the value of a in four ways. One way is to abbreviate the statement "Expression *x* has the same value as a." by just writing *x*, without saying anything about it. Whenever we just write a binary expression, we mean that it has the same value as a (expresses the same value that a expresses). For example, instead of saying "Expression a = a has the same value as a." we just say "a = a". (Remember that truth is just one of the binary values in just one of the applications.)

Ternary algebra

Between the two values represented by a and ⊥, we now consider another value, represented by 0 (pronounced "zero"). Ternary algebra can be applied to anything that comes in three kinds. In one application, the three expressions a, 0, and ⊥ represent the values "yes", "maybe", and "no". In another, they represent the values "large", "medium", and "small". An assignment of values to

variables that gives an expression the value 0 is called a "root" of the expression.

The expressions of ternary algebra, called "ternary expressions", include all those of binary algebra. To determine the value of these ternary expressions, we extend the value tables.

x	⊤	0	⊥
$-x$	⊥	0	⊤

xy	⊤⊤	⊤0	⊤⊥	0⊤	00	0⊥	⊥⊤	⊥0	⊥⊥
$x=y$	⊤	⊥	⊥	⊥	⊤	⊥	⊥	⊥	⊤
$x\neq y$	⊥	⊤	⊤	⊤	⊥	⊤	⊤	⊤	⊥
$x<y$	⊥	⊥	⊥	⊤	⊥	⊥	⊤	⊤	⊥
$x>y$	⊥	⊤	⊤	⊥	⊥	⊤	⊥	⊥	⊥
$x\leq y$	⊤	⊥	⊥	⊤	⊤	⊥	⊤	⊤	⊤
$x\geq y$	⊤	⊤	⊤	⊥	⊤	⊤	⊥	⊥	⊤
$x\wedge y$	⊤	0	⊥	0	0	⊥	⊥	⊥	⊥
$x\vee y$	⊤	⊤	⊤	⊤	0	0	⊤	0	⊥
$x\triangle y$	⊥	0	⊤	0	0	⊤	⊤	⊤	⊤
$x\triangledown y$	⊥	⊥	⊥	⊥	0	0	⊥	0	⊤

When the variables have binary values, each expression has the same value as it had in binary algebra; in that sense, we have extended binary algebra to ternary algebra in a consistent way. All our future extensions will likewise be consistent. The expression $x = -x$ has no solution in binary algebra because both assignments of binary values give it the value ⊥; in ternary algebra it has solution 0.

Common laws

I have introduced binary and ternary expressions, and mentioned expressions of four or more values. I am about to introduce numbers, bunches, sets, strings, lists, and functions by saying how to write them and giving their laws. There are some laws of binary algebra that are not laws of any other algebra; for example,

$(x\leq y) = -x \vee y$	material order
$((x=y)=z) = (x=(y=z))$	associative
$((x+y)+z) = (x+(y+z))$	associative

The next two expressions are laws of binary algebra, and one of the four-valued algebras mentioned in the previous section, but not of any other algebra mentioned in this paper.

$x\vee -x$	excluded middle
$-(x\wedge -x)$	non contradiction

There are laws of some of our algebras that are not laws of binary algebra, but only because they employ symbols that are not symbols of binary algebra. Any law of any of our algebras that employs only the symbols of binary algebra is also a law of binary algebra.

There are many laws that are common to all of the algebras in this paper; for example,

$\perp \leq x \leq a$	extremes
$x\wedge\perp = \perp$	base
$x\vee a = a$	base
$x \perp = a$	base
$x\ a = \perp$	base
$x\wedge a = x$	identity
$x\vee\perp = x$	identity
$(x=a) = x$	identity
$(x+\perp) = x$	identity
$x = x$	reflexivity
$x \leq x$	reflexivity
$x \geq x$	reflexivity
$-(x<x)$	irreflexivity
$-(x>x)$	irreflexivity
$- -x = x$	double negation or self-inverse
$x\wedge x = x$	idempotence
$x\vee x = x$	idempotence
$(x=y) = (y=x)$	symmetry
$(x\neq y) = (y\neq x)$	symmetry
$x\wedge y = y\wedge x$	symmetry
$x\vee y = y\vee x$	symmetry
$x \mid y = y \mid x$	symmetry
$x - y = y - x$	symmetry
$-(x < y < x)$	antisymmetry
$-(x > y > x)$	antisymmetry
$-(x < y = x)$	exclusivity
$-(x > y = x)$	exclusivity
$(x\leq y) = (x<y) \vee (x=y)$	inclusivity
$(x\geq y) = (x>y) \vee (x=y)$	inclusivity
$(x>y) = (y<x)$	mirror
$(x\geq y) = (y\leq x)$	mirror
$(x<y) = (-x>-y)$	reflection
$(x\wedge y = x) = (x\leq y) = (y = x\vee y)$	connection
$x\wedge(x\vee y) = x$	absorption
$x\vee(x\wedge y) = x$	absorption
$-(x=y) = (-x \neq -y)$	duality
$-(x+y) = (-x = -y)$	duality
$-(x<y) = (-x \leq -y)$	duality
$-(x\leq y) = (-x < -y)$	duality
$-(x>y) = (-x \geq -y)$	duality
$-(x\geq y) = (-x > -y)$	duality
$-(x\wedge y) = -x \vee -y$	duality
$-(x\vee y) = -x \wedge -y$	duality
$-(x\ y) = -x - y$	duality
$-(x\ y) = -x - y$	duality
$x \wedge (x\leq y) \leq y$	modus ponens
$(x+y) = -(x=y)$	unequality
$x\ y = -(x\wedge y)$	neg min
$x\ y = -(x\vee y)$	neg max
$(x\wedge y)\wedge z = x\wedge(y\wedge z)$	associativity
$(x\vee y)\vee z = x\vee(y\vee z)$	associativity
$(x = y = z) \leq (x=z)$	transitivity
$(x < y < z) \leq (x<z)$	transitivity
$(x > y > z) \leq (x>z)$	transitivity
$(x \leq y \leq z) \leq (x\leq z)$	transitivity
$(x \geq y \geq z) \leq (x\geq z)$	transitivity
$x\wedge y \leq y \leq y\vee z$	specialization and generalization
$x\wedge(y\vee z) = (x\wedge y) \vee (x\wedge z)$	distribution or factoring
$x\vee(y\wedge z) = (x\vee y) \wedge (x\vee z)$	distribution or factoring
$(x \leq y\wedge z) = (x\leq y) \wedge (x\leq z)$	distribution or factoring
$(x \leq y\vee z) \geq (x\leq y) \vee (x\leq z)$	distribution or factoring
$(x\wedge y \leq z) \geq (x\leq z) \vee (y\leq z)$	antidistribution
$(x\vee y \leq z) = (x\leq z) \wedge (y\leq z)$	antidistribution
$(w\wedge x) \vee (y\wedge z) \leq (w\vee y) \wedge (x\vee z)$	

$x<\mathrm{a}>y = x$	base
$x<\perp>y = y$	base
$-(x<y>z) = -x<y>-z$	distribution or factoring

It is an interesting mathematical exercise to find a minimal set of laws for an algebra. But those who wish to use the algebra need to know many laws, and to them minimality is of no concern. In this paper, no attention has been paid to minimality.

Number algebra

I now introduce infinitely many values between a and ⊥. Here are some of them.

⊥ –3 –2 –1 0 1 2 3 a

All operators apply to all values.

The expressions of number algebra are called "number expressions". They can be used to represent anything that comes in quantities, such as apples and water (a represents an infinite quantity, and ⊥ represents an infinite deficit). Expressions are formed as follows. any sequence of one or more decimal digits, such as 5296 any of the ways of forming an expression presented previously, such as

–5296 or 5296∧375 or 5297=375

$x+y$	" x plus y "
$x-y$	" x minus y "
$x\times y$	" x times y "
x/y	" x divided by y ", " x over y "
xy	" x to the power y "

Anyone is welcome to invent new expressions and add them to the list.

Now that we have new expressions, we assign some of them the same value as †. In these laws, d is a sequence of digits.

$d0+1 = d1$	counting
$d1+1 = d2$	counting
$d2+1 = d3$	counting
$d3+1 = d4$	counting
$d4+1 = d5$	counting
$d5+1 = d6$	counting
$d6+1 = d7$	counting
$d7+1 = d8$	counting
$d8+1 = d9$	counting
$d9+1 = (d+1)0$	counting
$x+0 = x$	identity
$x+y = y+x$	symmetry
$x+(y+z) = (x+y)+z$	associativity
$(\perp<x<\mathrm{a}) \le ((x+y = x+z) = (y=z))$	cancellation
$(\perp<x) \le (\mathrm{a}+x = \mathrm{a})$	absorption
$(x<\mathrm{a}) \le (\perp+x = \perp)$	absorption
$x + y\wedge z = (x+y) \wedge (x+z)$	distributivity or factoring
$x + y\vee z = (x+y) \vee (x+z)$	distributivity or factoring
$x + y\ z = (x-y) \vee (x-z)$	
$x + y\ z = (x-y) \wedge (x-z)$	
$x + (y<z>w) = x+y<z>x+w$	distributivity or factoring
$-x = 0 - x$	negation
$-(x+y) = -x + -y$	distributivity or factoring
$-(x-y) = -x - -y$	distributivity or factoring
$-(x\times y) = (-x)\times y$	associativity
$-(x/y) = (-x)/y$	associativity
$x-y = -(y-x)$	antisymmetry
$x-y = x + -y$	
$x + (y - z) = (x + y) - z$	associativity
$(\perp<x<\mathrm{a}) \le ((x-y = x-z) = (y=z))$	cancellation
$(\perp<x<\mathrm{a}) \le (x-x = 0)$	inverse
$(x<\mathrm{a}) \le (\mathrm{a}-x = \mathrm{a})$	absorption
$(\perp<x) \le (\perp-x = \perp)$	absorption
$(\perp<x<\mathrm{a}) \le (x\times 0 = 0)$	base
$x\times 1 = x$	identity
$x\times y = y\times x$	symmetry
$x\times(y+z) = x\times y + x\times z$	distributivity or factoring
$x\times(y\times z) = (x\times y)\times z$	associativity
$(\perp<x<\mathrm{a}) \wedge (x+0) \le ((x\times y = x\times z) = (y=z))$	cancellation
$(0<x) \le (x\times \mathrm{a} = \mathrm{a})$	absorption
$(0<x) \le (x\times \perp = \mathrm{a})$	absorption
$x/1 = x$	identity
$(\perp<x<\mathrm{a}) \wedge (x+0) \le (x/x = 1)$	inverse
$x\times(y/z) = (x\times y)/z = x/(z/y)$	multiplication-division
$(y+0) \le (x/(y/z) = x/(y\times z))$	multiplication-division
$(\perp<x<\mathrm{a}) \le (x/\mathrm{a} = 0 = x/\perp)$	annihilation
$(\perp<x<\mathrm{a}) \le (x0 = 1)$	base
$x1 = x$	identity
$xy+z = xy\times xz$	exponents
$xy\times z = (xy)z$	exponents
$\perp<0<1<\mathrm{a}$	direction
$(\perp<x<\mathrm{a}) \le ((x+y < x+z) = (y<z))$	cancellation, translation
$(0<x<\mathrm{a}) \le ((x\times y < x\times z) = (y<z))$	cancellation, scale
$(x<y) \vee (x=y) \vee (x>y)$	trichotomy

Calculation

Given an expression, we might find a simpler expression with the same value. For example,

$x\times(z+1) - y\times(z-1) - z\times(x-y)$	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
$= x+y$	

The entire five lines (without the hints that appear to the right) form one binary expression meaning the same as

$(x\times(z+1) - y\times(z-1) - z\times(x-y) = (x\times z + x\times 1) - (y\times z - y\times 1) - (z\times x - z\times y))$
$\wedge ((x\times z + x\times 1) - (y\times z - y\times 1) - (z\times x - z\times y) = x\times z + x - y\times z + y - z\times x + z\times y)$

$\wedge\ (x\times z + x - y\times z + y - z\times x + z\times y = x + y + (x\times z - x\times z) + (y\times z - y\times z))$
$\wedge\ (x + y + (x\times z - x\times z) + (y\times z - y\times z) = x+y)$

By simply writing it, we are saying that it has the same value as a. The hint "distribute" is intended to make it clear that
$x\times(z+1) - y\times(z-1) - z\times(x-y) = (x\times z + x\times 1) - (y\times z - y\times 1) - (z\times x - z\times y)$

is a; the hint "unity and double negation" is intended to make it clear that
$(x\times z + x\times 1) - (y\times z - y\times 1) - (z\times x - z\times y) = x\times z + x - y\times z + y - z\times x + z\times y$

is a; and so on. By the transitivity of = and the Consistency Rule we see that
$x\times(z+1) - y\times(z-1) - z\times(x-y) = x+y$
is a, and so $x\times(z+1) - y\times(z-1) - z\times(x-y)$ and $x+y$ have the same value.

We can use operators other than = down the left side of a calculation, even a mixture, as long as there is transitivity. For example, if x is a real-valued variable,

$x\times(x + 2)$	distribute
$= x2 + 2\times x$	identity and zero
$= x2 + 2\times x + 1 - 1$	factor
$= (x+1)2 - 1$	a square is nonnegative
≥ -1	

tells us that $x\times(x+2) \geq -1$ is a. The level of hint depends on the knowledge of the intended audience.
I have presented an algebra that unifies numbers with booleans, types with values, and function spaces with functions. There is no loss of structure, just loss of duplication. This is mathematics by design. Like any design, it is neither right nor wrong; the criteria for judging it are usefulness and elegance.

References

1. Grundy J. Transformational Hierarchical Reasoning. The Computer Journal 1996;39(4):291-302.
2. Hehner ECR. From Boolean Algebra to Unified Algebra, the Mathematical Intelligencer, Springer 2004;26(2):3-19. www.cs.toronto.edu/~hehner/BAUA.pdf
3. Hehner ECR. A Practical Theory of Programming, Springer 1993. www.cs.toronto.edu/~hehner/aPToP
4. Hoare CAR. A Couple of Novelties in the Propositional Calculus, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 1985;31(2):173-8.