An Operators Manual for Algebra

How to use algebra

2023-07-25

James B. Wilson
Department of Mathematics
Colorado State University

James.Wilson@ColoState.Edu

James.Wilson@ColoState.Edu

Contents

What is algebra? 1

Algebra is the study of equations, for the most part equations involving variables. That is because applications have unknowns and if the the shape of the equation can tell us anything about the options to solve it we shall want to take advantage of this. Witness how we divide $ax^2 + bx + c = 0$ into solutions that are real or complex based on $b^2 - 4ac \geq 0$.

You probably already know every color, shape, and pattern of equation you will ever need. Compare these two equations

$$x^2 + y^2 \equiv 0 \pmod{541} \qquad \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0.$$

These equations concern entirely unrelated context, for example 0 on the right is a whole number, whereas 0 on the left is the function $0(x, y) = 0.0$ Yet, the similarities as equations shine through. This is because 0, 1, $+$, and powers of 2 are abstractions. The similarities we are observe are an exploration of a single equation

$$x^2 + y^2 = 0.$$

In fact, even the equality was an abstraction which could vary from context to context. With such flexibility a small number of symbols and their grammar are enough to capture the huge variety of equations we encounter in real life.

Now since every symbol in an equation is a variable we have new powers to solve equations. Look to the humble

$$x^2 + 1 = 0.$$

By our own view, right now this is nothing but variables, so it means nothing to solve this. But, drop this into a context such as decimal numbers $\mathbb{R}$ and the understanding is to replace $1 := 1.0$, $0 := 0.0$, $+$ is substituted for addition of decimals, and square is by multiplying decimals. Equality now means two equal decimals, or in practice two decimal numbers that are close enough to be considered equal. The only remaining unknown is x, but as everyone knows, we wont find a solution as no square decimal is -1.

The power of variable everything is that we are not stuck with the real numbers. Lets replace everything with complex numbers $\mathbb{C}$. Substitute $0 := 0+0i$, $1 = 1+0i$, $(a+bi)+(c+di) := (a+b)+(c+d)i$, and $(a+bi)^2 = (a^2-b^2)+2abi$. Now we find $\pm i$ are the solutions. Solutions do exist! Why stop here? Try quaternions, or (2×2)-matrices. These each have addition, 0, 1, and squares. Now we learn there can be infinitely many solutions to that equation. This is the method of algebra: find all the values that can be substituted into an equation to see what makes solutions possible and how they behave. Learn enough by this process and we can begin to predict if solutions are to be expected and fathom algorithms to find them when they do exist. When the solutions become infinite we find ways to parameterize them with smaller data such a basis.

Don't forget equality is variable too! Suppose we wanted to solve $x^{541}+x+1=0$ using only integers. Replace equality with $\equiv$ modulo 2 and ask for a whole number x that solves

$$x^{541}+x+1 \equiv 0 \pmod 2.$$

All integers in this equality become either 0 or 1, but neither will solve this equation. By varying equality we confirm there are no solutions.

The power of algebra is that every symbol in an equation is a variable, especially the equals sign.

What is +? 2

One day + means to add natural numbers, the next day polynomials, later matrices. You can even add colors “Yellow=Blue+Green”. When you program you learn to add strings

```
In [1]  print "Algebra " + " is " + " computation"

Out [1] Algebra is computation
```

The + is in fact a variable stand in for what we call *binary operator* or *bi-valent operator*, as it takes in a pair, according to the grammar $\square + \square$. The valence and the grammar of an operator comprise its *signature*.

Unlike these notes, addition should only be used when it is grammatically correct e.g. $2+3$ rather than $+23$. Think of this like any other language where there could be a dialect that evolves the operator's grammar and lexicon. A program to add two lists could get away with the following linguistic drift:

```
In [2]  cat [3,1,4] [1,5,9]
        [3,1,4] + [1,5,9]

Out [2] [3,1,4,1,5,9]
        [4,6,13]
```

Obviously using `cat` avoided confusion with the later + concept and was the better choice. Challenge yourself to see both as addition and you will find addition everywhere.

Since we are evolving we my as well permit multiplication as a binary operator symbol, changing the signature to $\square \cdot \square$, i.e. $2 \cdot 4$; or $\square\square$, e.g. xy. Avoid $\square \times \square$, we need that symbol elsewhere. Addition is held to high standards in algebra (that it will evolve into linear algebra). So when you are considering a binary operation with few if any good properties, use a multiplication inspired notation instead.

Valence 1 operators include the negative sign $-\square$ to create -2. Programming languages add several others such as `++i`, `--i` which are said to *increment* or *decrement* the counter i (change it by ± 1). Programs also exploit a ternary (valence 3) operator:

```
if (...)then (...)else (...)
```

You may find that with shorter syntax like `n!=0 ? m/n : error`. If your interested look into *variadic* operators to see how far this idea goes.

Operators that generate

Actually incrementing is more a definition than it is a special case of adding 1. A natural number is either 0, or a successor $S(k)$ to another natural number k. Often this is expressed as a definition where | stands for separating cases, for example:

$$\mathbb{N} := 0 \mid S(k)$$

So 0 is "zero", and 1 is just a symbol representing $S(k)$, $2 = S(S(0))$ and so on. Replace S's with tally marks we recover childhood counting:

$$0 := \_\_, \; 1 := |, \; 2 := ||, \; 3 := |||, \; 4 := ||||, \; 5 := \text{𝍸}, \ldots$$

The point is, the successors are not so much a function moving around the numbers we have, it actually is a producer of numbers.

Perhaps because it is so primitive, this is an idea we can imitate to create more meaningful values, like a string of characters in an alphabet `['a','b',...,'z']`.

```
data String = Empty | Prepend( head, tail)
```

Some readers might relate to a different dialect of programming such as the following

```
class String
    case Empty extends List
    case Prepend( head, tail) extends List
```

The head here caries around what we put in the list and the tail is what comes next in the list. Observe the similarities:

$$2 := S(S(0)) \qquad (\mathbb{N})$$

$$\texttt{"me"} := \texttt{Prepend('m',Prepend('e',Empty))} \qquad (\text{String})$$

The left-hand sides are merely notation for what the data really is on the right. Both the successor and the `Prepend` are operators that generate new values. So part of algebra is to generate new data; so, it is no wonder that is closely connections to computation.

Generated operators

We can take this the idea of generating further, for example, using the unary operator, successor, prepend, etc., and have them generate binary operators.

$$m + n := \begin{cases} n & m = 0 \\ S(n+k) & m = S(k) \end{cases}$$

So does $2 + 4 = 6$? We can test this out.

$$\begin{aligned} || + |||| &= | \left(| + |||| \right) \\ &= | \left(| \left(\_\_ + |||| \right) \right) \\ &= | \left(| \; |||| \right) \\ &= | \; \text{卌}. \end{aligned}$$

Try this for strings

$$s + t := \begin{cases} s & t = \texttt{Empty} \\ \texttt{Prepend}(x, n + tail) & t = \texttt{Prepend}(x, tail) \end{cases}$$

What is `"awe"+"some"`?

While no one will seriously add integers as a tally, knowing that it can be done establishes a pattern which can be exported to other context with meaningful new structure. Just notice $3+4 = 4+3$ but not so with adding strings. These siblings have their own

personalities, and it this might even help us recognize that while $3+4$ does equal $4+3$ it might be for somewhat subtle reasons.

What about multiplication? Isn't is just this:

$$m \cdot n := \overbrace{n + \cdots + n}^{m}.$$

That is nice, but seems to leave us to figure out missing parenthesis or set aside time to prove they don't matter. I am in the mood to continue making things rather than study them. Lets repeat what we have done.

$$m \cdot n := \begin{cases} 0 & m = 0 \\ k \cdot n + n & m = S(k). \end{cases}$$

It works, but check it out for yourself. And for strings what might we get?

$$s \cdot t := \begin{cases} 0 & s = \texttt{Empty} \\ \texttt{Prepend}(x, k \cdot n) + n & s = \texttt{Prepend}(x, tail) \end{cases}$$

What is "Mua"·"Ha"? You can carry on to make exponents and more. If you do you may find Knuth arrow notation helpful on your journey, look it up.

Operators measuring defects

Algebraist spend a lot of time worried about misbehaving operators causing them to generate new operators that spot the flaws. If we can add and multiply then we can make the following operators as well.

$$[a, b] = ab - ba \qquad \text{(Commutator)}$$
$$(a, b, c) = a(bc) - (ab)c \qquad \text{(Associator)}$$

Commutative algebra requires $[a, b] = 0$ while associative algebra needs $(a, b, c) = 0$. For example matrices fail to be commutative algebra but are associative. Replace the role of multiplication of matrices with $[a, b]$ and ask for it's associate, i.e.

$$(a, b, c)_{[,]} = [a, [b, c]] - [[a, b], c]$$

and we no longer get associative nor commutative algebra. This are structures known as Lie algebras. While not associative, because they are based originally on matrix products that are associative we can stumble eventually upon a graceful alternative

$$0 = [a, a] \qquad \text{(Alternating)}$$
$$0 = [a, [b, c]] + [b, [c, a]] + [c, [a, b]]. \qquad \text{(Jacobi)}$$

So the problem is not getting worse, at least we wont be needing to look into some valence 4 operators as defects. Sabinin algebra studies how defects in operators pile up or die off.

Keep in mind requiring that $[a, b] = 0$ or $(a, b, c) = 0$ is an equation. Like any equation it has limited solutions. By that reasoning, most of algebra wont be behave nicely.

Operators forced into service

Sometimes you have to be clever, even lucky, to guess an operator. In famous history, Heisenberg escaped to an island from Copenhagen to avoid hay-fever, or an over zealous host Niels Bohr. Able to think clearly he summarized what he knew: a particle ψ could take possibly many states, maybe spin up $\uparrow$ or spin down $\downarrow$, so it record this as linear combination it as a linear combination $|\psi\rangle = \alpha|\uparrow\rangle + \beta|\downarrow\rangle$ where in vector space with $\{\uparrow, \downarrow\}$ as a basis.

somewhat famously was having difficulty with hay-fever in Copenhagen and was stuck with quantum puzzles who could not solve. Clearing his head on an island he conjured the view that particles ψ could be viewed as linear combinations of all the states the could o $\langle\psi| = \alpha\langle$ consisting of the probabilities of each state. recognized matrices capture all the states of a quantum system and to compute the next event involv

Properties

You can add music, well at least you can play two songs at the same time. Is that again music? Probably it is safer to say we can add sound, and some sound is music. This is thinking like an algebraist, this is clarifying that addition has a context, sound in

this case, and the operations can be carried out without leaving that context. We say that:

> "Sound is closed under addition."

You needn't be too strict about the context. When you add a list of length 3 to a list of length 3 it gets to a list of length 6, not 3. Zoom out the context of lists of a general length and you regain closure. This may not be enough. When we divide we avoid division by 0. When we subtract natural numbers $m - n$ we need $m \geq n$. When we compose functions we need the domain of the second to contain the image of the first. You might be in a place to fix this, for example add $\pm\infty$ to define as $x/0$ or negative numbers to account for $3 - 5$. In composable functions f and g can compose as relations:

$$(g \circledcirc f)(x) := \{z \mid \exists y, y = f(x), z = g(y)\}.$$

When doing algebra with such operators we nearly always spend our time explore independent cases, most of which are roughly speaking "errors". The better idea is to acknowledge we don't really want to divide by 0, produce negatives when we are studying positive numbers, and we shouldn't compose non-composable functions.

Further operators

These examples lean on some addition pre-existing somewhere.

Other operators

To enforce a philosophy such as that some algebra is closed to operators Such a philosophy is one

To get the details rolling in earnest we must know the application. The world around us partly interacts with computers so lets assume this is a uniform assumption: any problem in algebra that I want to state should be expressible to a computer. That computer might not be able to handle it but its hard to imagine

an equation we need to consider where the equation itself cannot be stored in a computer.

That is one philosophy, a philosophy where addition is an abstraction. That word makes some bristle. It reminds them of polarizing art installations or refrains from the bar room rants between pure and applied thinkers. Abstract, for for those who reason, means to limit argument to specified attributes. Thats every type of math and a good chunk of science so it shouldn't raise dismay. *Raising the level of abstraction* is then just the declaration the we're about to gather the instance we have and ma

Perhaps you feel some one owes you a foundation for addition, a place you derive what addition really means before you take to making it show up everywhere else. You can count on it, literally.

$$\mathbb{N} = 0|S\ k.$$

How do we do algebra? 3

This leaves us with the job of making those data which can be substituted into equations. This means firstly new numbers that, like the decimals, complex numbers, polynomials or matrices; can give a meaning to things like 0 and 1 and eventually the answers x we seek in solving applications. Secondly we need to find what works for the operations, the sums, squares, products, and etcetera. Last and most important in the method of algebra is to invent the possible substitutes for equality, what algebraist call *congruences*.

Over 2 centuries the methods have evolved and with it notation and emphasis to the point where the three roles just articulated may not be recognizable. This is where it becomes necessary to add in constraints: a family of problems you need solved that guide you to the algebra you need. But when we pause to see the similarities we can carry forward a far great number of algebraic lessons than we can by concentrating only on the best tools for individual examples. That will be our perspective. To further constrain the study we invite in applications that constrain the questions to important families the ones which might solve your problem or lead you to the algebra the may one day define you.

Formulas & Equations 4

Lets get precise about equations. They have a left hand side and right hand side. But the items that can occur on the left can also occur on the right so we could define any one side and be done. The equations worth study are those involving variables, more precisely variable and operations since and equation $x \cdot y = 1$ is infinitely more interesting than $x = 1$. So we start out with what are called formulas, but you would do well to think of these as generalized polynomials, only we might involve more operations than simply those found in common polynomials. As a byproduct we will re-invent induction in a form that is used everywhere in in the design of software data types.

Equality 5

One day + means to add natural numbers, the next day polynomials, or matrices. You can even add colors "Yellow=Blue+Green". You can add music, well at least you can play two songs at the same time and decide if you want to call it music. In fact

The + is there as a notation to explain there is a pattern at play that you have seen before. All additions come in pairs for instance. Its the signature of addition to write the pair as $x+y$. Beyond that there are many reasonable assumptions you might make with the presence of +. You probably suspect that any + can be done in any order: Green+Blue also equals Yellow, 2+3 is the same as 3+2. And I bet you expect that parentheses don't matter either with a +.

To call them all addition suggests we see things in common and we can borrow experience from other numbers to guess at properties for the next system. When we are successful in abstracting the right properties we can turn a guess into a proof. For example, the typical situation these days is that the symbol + abstracts any process that turns a pair of numbers x and y into a new one, written $x+y$, along with the property that $x+y=y+x$ and $x+(y+z)=(x+y)+z$. Important abstractions get clever names: $x+y=y+x$ is the *commutative law*, like dance partners the variable move past each other but stay together. The second abstraction is the *associative law*, where the variables are stuck in a line but can talk to the one in front or behind them. But you know this already.

That is because of a a trick: *abstraction*, which means to study a subject by considering only some of its possible attributes. For example,

Yet if you open a text on algebra that isn't for high-schools you find instead page after page of Of course every algebraist knows this but it

since applications often need us to solve for one. Its simply not as important to study $2^2+2+1=7$ as it is to sol Of course we could study 2=2 and The method however is to of study is Start with $x^2+1=0$. What It has no real solutions.

The first lesson of algebra is that every symbol can be variable. For example imagine focussing on solving $x^2 = 3$.

So focus on solving $x^2 + x + 1 = 0$.

For example consider the equation $x^2 + x + 1 = 0$. What When you see $x + 3 = 5$

Functions 6

The first lesson of algebra is that everything you see is a variable. you can improve how we study equations

A function takes an input and use some process to determine an output, under the premise that this process is repeatable. So reading the time off a watch is not a function because time keeps changing, but converting time from hours to minutes is a function because every hour has 60 minutes. From that philosophical understanding two functions are obvious:

$$I(x) = x \qquad\qquad K_c(x) = c.$$

We call the first an *identity* function and the second is a *constant* function. This notation is misleading, technically I is a symbol which denotes some unknown process such that when applied to data ♣, its output, often denoted $I(♣)$, is given input data unchanged. So $I(♣) = ♣$ is a judgement we can make about an identity function not a program.

There is a bit of implicit information in our use of parenthesis and what they mean. Traditionally I and K are the functions. An input ♣ passed to a function I yields an output denoted $I(♣)$. In the case of the identity function I does nothing to modify the input so we can judge $I(♣) = ♣$. Since this applies for any input we can abstract over the inputs by replacing their role with a variable x and write $I(x) = x$ It is not a definition of I so much as a consequence. Of course we could use the outcome rule to conjure a perspective algorithm to perform the function. When we do this we often insert some extra language like “define I(x)=x” or use the Walrus notation $I(x) := x$. In programs words like define are abbreviated. For example, the following two programs satisfy the identity function rule without following the same process.

```
def doNothing(x) = x
def doNothingUseful(x) = compute 200! then return x
```

Much of the feasibility of algebra comes down to appreciating different processes that achieve the same overall calculations.

nothing in the statement $I(x) = x$ prevents the function from reading in an input, taking a trip to mars and returning, then outputting x. So the description $I(x) = x$ is not so much a program but an fact

what we write. Writing $I(x) = x$ is a judgement we can make after using the function I. For example, the identity function applied to 3, often written $I(3)$, yields 3. Indeed $I(3) = 3$. In fact if we want a function J that never changes the input then all such J will have the quality that $J(x) = x$ in the end. Of course nothing prevents J form first move the data around from place to place, Curiously this rule seems to So without even considering an algorithm to act Back tracking from the conclu

a function of x meaning x is the variable for which we will substitutes other quantities, $y(3) = 3$ whereas $\acute{y}(3) = 2$. In fact $\acute{y}_2 = 2$ is part of a family of functions which we can describe as general *constant functions* denoted

Composing means to send the output of one function to the input of another so we could do this with our identity and constant functions

These are not the same functions of course.

That they are functions is more like a rule than a fact. When you need to formalize this a bit more you can start by saying you have an alphabet of symbols containing I, K. You

If you needed to formalize this a bit more you could say I is a symbol and there is rule $Thatis$,I(x)$justmeans$

writing $I(x) = x$ is notation for "I applied to data x returns x" and because it would take some circular reasoning to explain what "I(x)=x" means if not to operate as function that changes nothing. We might like to compose functions, for instance IK_c or In fact this brings up a problem of what it

6.1