Expression Trees

These trees are pictures which show the structure of math expressions

- Trees form a clearer picture of the Algebra 1 properties

Properties
associativity
commutivity
identity
(inverse)
distributive

- Trees eliminate the order of operations, preventing many common mistakes

numbers

3 51 height

-1 10.50

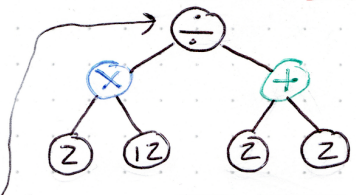
functions

Each leg is an input

expressions

Each input is filled with a number or expression.

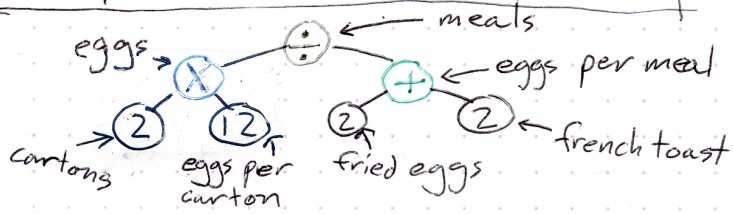
There is just one top function.



$(2 \times 12) \div (2 + 2)$

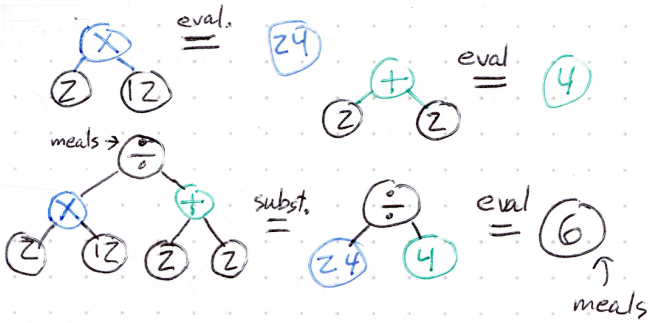
You form expressions by joining values together with functions. This means each ○ in the expression tree is a value we can name. Names help you understand what the expression is saying.

In english: We're cooking fried eggs and french toast meals. Each meal needs 2 fried eggs and 2 eggs for the french toast. We have 2 cartons of a dozen eggs. How many meals can we make?

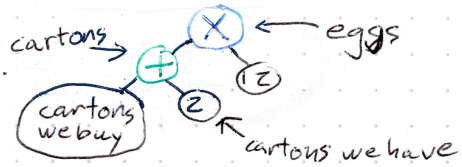


Algebra is about rearranging expressions into equivalent exprs to simplify the expr or discover properties and constraints of the expr

The primary rule of algebra is you can substitute values in the expr with equal values. On the right, we evaluate the expression by substituting each function whose inputs are numbers with a number



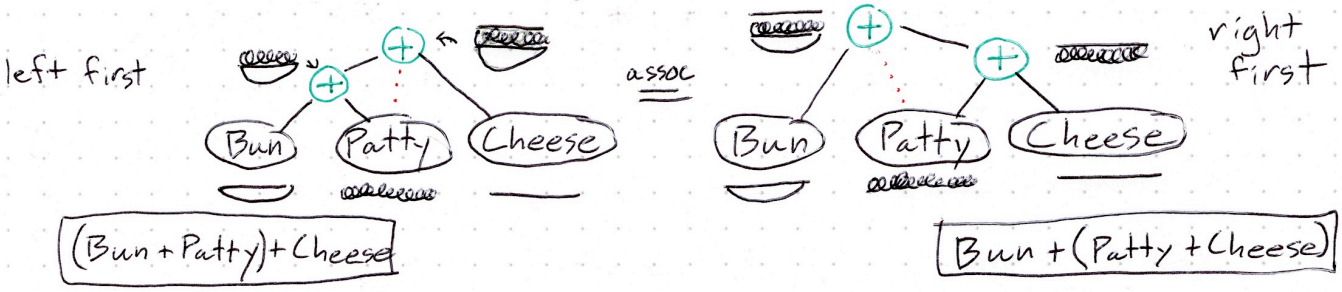
Algebra's complexity comes from variables. These are inputs in our expr we can change. For example, we can buy more cartons of eggs. The expression on the right says how to calculate the number of meals from the number of cartons we buy.



You can evaluate your expr if it has variables. You must first fix the variable to a single value. Rewriting an expr with variables uses the following properties of these functions.

Associativity

A function is associative if:
- when joining any 3 values with the function,
- joining the middle value with the left first
  is always equal to joining the middle with the right first.

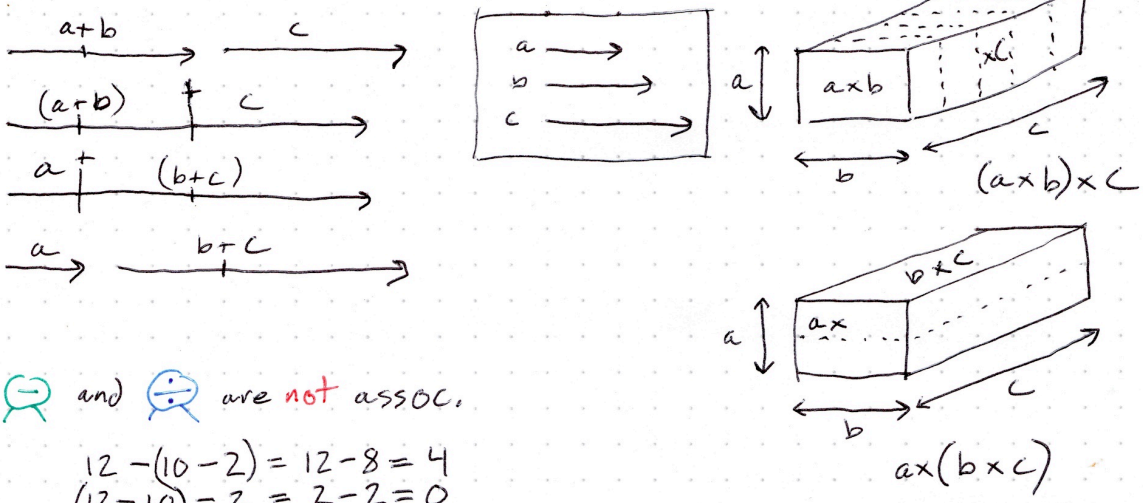


(Bun + Patty) + Cheese

Bun + (Patty + Cheese)

This property is so-named because the middle value may associate with either the left or right.

The ⊕ and ⊗ functions are assoc. Proofs:



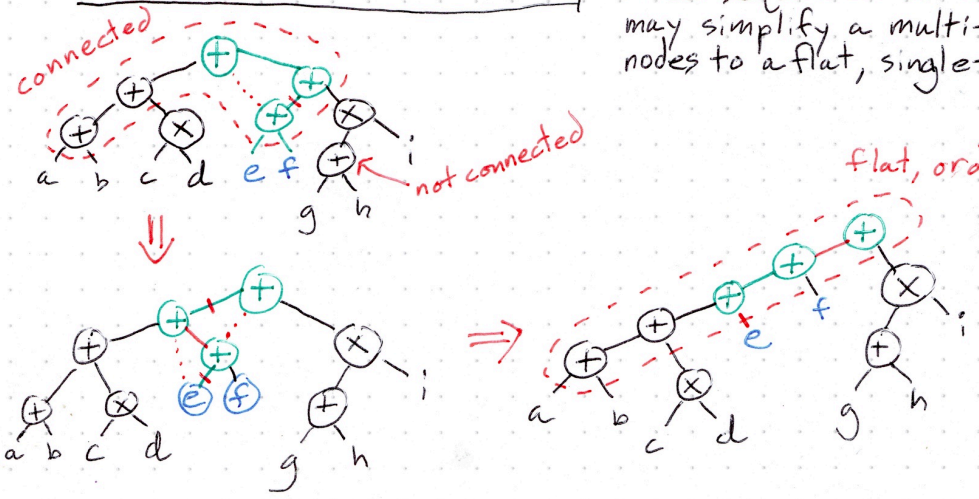
⊖ and ⊘ are not assoc.

$12 - (10 - 2) = 12 - 8 = 4$
$(12 - 10) - 2 = 2 - 2 = 0$

$12 \div (6 \div 2) = 12 \div 3 = 4$
$(12 \div 6) \div 2 = 2 \div 2 = 1$

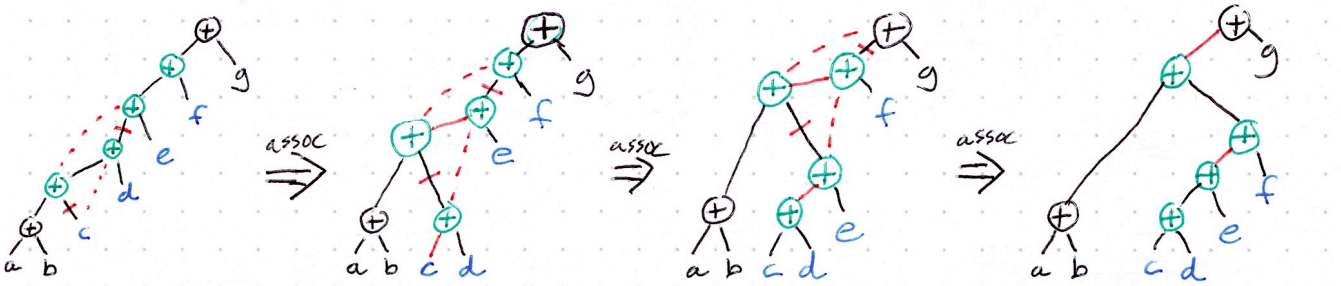
Assoc ⇒ Flattenable

A consequence of associativity is you may simplify a multi-level group of connected nodes to a flat, single-level ordered list.



Assoc ⇒ Groupable

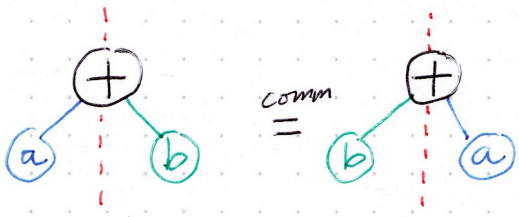
Another consequence of assoc is that you can group any middle part of an ordered list.



$a+b+c+d+e+f+g = a+b+(c+d)+e+f+g = a+b+(c+d+e)+f+g = a+b+(c+d+e+f)+g$

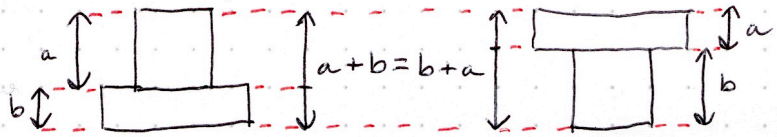
Commutivity

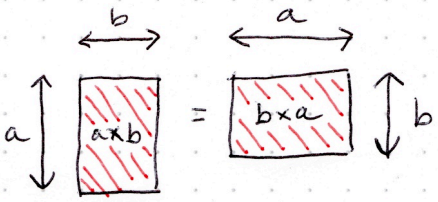
A function is commutative if:
- for any 2 inputs
- swapping the inputs gives the same output



This property is so-named because commuting means 'exchange'. When you commute from home to school, you are changing your location.

The ⊕ and ⊗ functions are commutative.





The ⊖ and ⊘ functions are not comm.

$4 \div 2 = 2$ $\quad$ $4 - 2 = 2$

$2 \div 4 = 1/2$ $\quad$ $2 - 4 = -2$

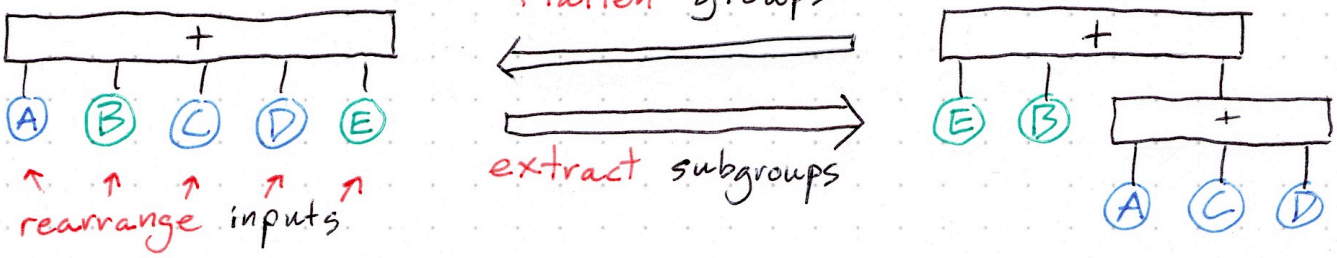
Assoc + Comm ⇒ Reorderable

A very useful consequence of the assoc and comm properties is that you can move values anywhere in the list. We can visualize the list ■ of inputs as an unordered bag of inputs.

... and so on until we've moved (a) where we want it

Comm + Assoc Summary

I like picturing connected groups of ⊕ or ⊗ as a single bag. There are 3 ways to rearrange bags

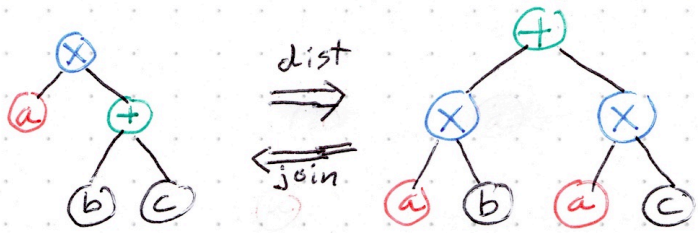


a+b+c+d+e

e+b+(a+c+d)

Distribution

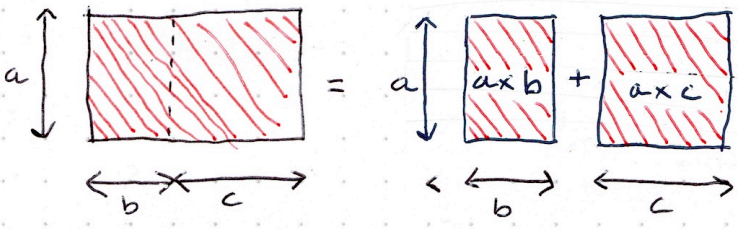
This property lets you swap ⊕ and ⊗. ⊗ distributes an input over ⊕ to each input.



a×(b+c)

(a×b) +(a×c)

Going the other direction is harder, but possible. You can join on a shared ⊕ input over ⊗



Distributing ⊗ over 2 ⊕ inputs gives us the awkward FOIL mnemonic. I prefer the more general rule of every pair of a left and right input.