Mathematical Foundations for the Common Core
A Course for Middle and Secondary Teachers

James J. Madden
Department of Mathematics, Louisiana State University

Topic: Expressions and Trees

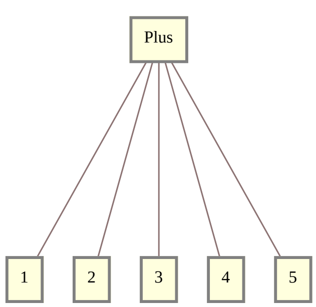
An expression is a written record of a computation. On page 62 of the *Common Core Standards*, a more elaborate account is given, which points out that the computations may start with numbers or with symbols: adding 3 to 10 is a computation, but so is adding 3 to x, if x is a number. Furthermore what is meant by a *computation* might be as simple as adding or multiplying, but taking the square root of a number or taking a trigonometric function of a number or stacking numbers in repeated exponents (as in $x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}$) is also a computation. Beyond this, computations can be strung together or combined with further computations to make complex, multistep computations. When these things are considered, it is clear that we have not yet provided a definition (in the precise mathematical sense) of the concept of an expression. We will get to that.

Let us look at some examples. Here are 6 different expressions. Each one uses each of the numbers from 1 to 5 exactly once and records a way of adding and/or multiplying these numbers.

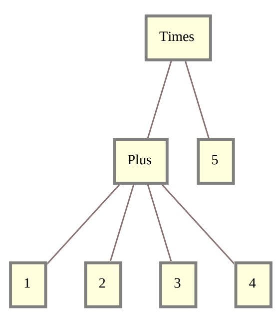
a) $1 + 2 + 3 + 4 + 5$
b) $(1 + 2 + 3 + 4)5$
c) $((1 + 2)3 + 4)5$
d) $(1 + 2)3 + (4 + 5)$
e) $1 + 2 + 3(4 + 5)$
f) $(1 + 2)(3)(4)(5)$

Expressions have structure. The structure of an expression can be made clear using a *tree diagram*:

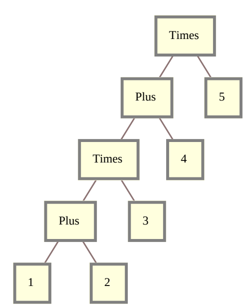
$a)$ $1 + 2 + 3 + 4 + 5$



$b)$ $(1 + 2 + 3 + 4)5$



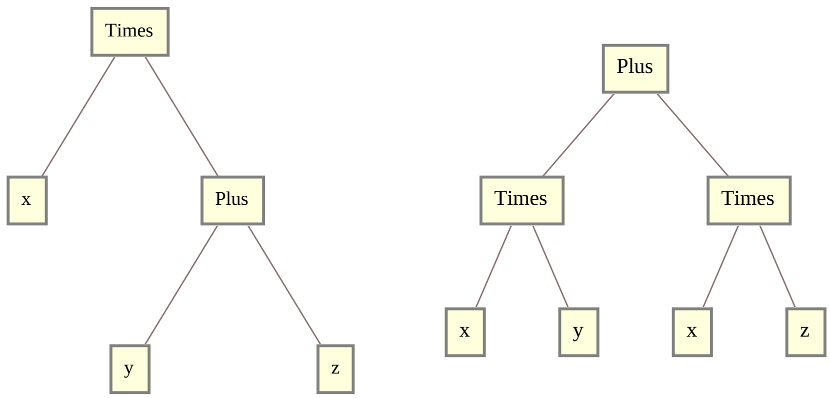
$c)$ $((1 + 2)3 + 4)5$



The expressions that we have been talking about do not have any variables in them. We call expressions such as these *numerical expressions* or *arithmetical expressions*. Expressions such at $x + 1$ or $a\,x^2 + b\,x + c$ that have variables in them are called *algebraic expressions*.

The distributive law is a rule for transforming expressions:

$$x\,(y+z) \qquad = \qquad x\,y + x\,z$$



Complexity of Expressions

In the last lecture, we observed that an arithmetic expression is simplified by performing the operations between numbers first. The effect of this is to work from the bottom of the tree, pulling the leaves into the nodes directly above them. In an algebraic expression, it may be impossible to make a modification that eliminates an operation: $x+y$ cannot be written in a simpler form. If we measure the complexity of an algebraic expression by the number of operation symbols in it, then many expressions just do not simplify. However, there are modifications that reduce a different kind of complexity.

Consider the following example:

$$\begin{aligned} x + x\,(x + x\,(x + x^2)) &= x + x\,(x + x^2 + x^3) \\ &= x + x^2 + x^3 + x^4 \end{aligned}$$

Including the multiplications implicit in the exponents, there are 6 operations in the first expression but 9 in the last. In terms of *alternations* between additions and multiplications, however, the last expression is much simpler. To explain what is meant, we need to analyze the structure of expressions more carefully.

Let us represent the sum of several inputs $x_1, x_2, \ldots, x_n$ by $\texttt{Plus}[x_1, x_2, \ldots, x_n]$. The inputs may be constants, variables, or the outputs of other operations. We will assume that none of the inputs to `Plus` are outputs of addition, since any `Plus`s that occur as heads of inputs to `Plus` are redundant (*e.g.*, $\texttt{Plus}[a, \texttt{Plus}[y, z]] = \texttt{Plus}[x, y, z]$). The analogue is true of multiplication. Let us represent the product of several inputs $x_1, x_2, \ldots, x_n$, which again may be constants, variables, or the outputs of any operations *other than multiplication*, by $\texttt{Times}[x_1, x_2, \ldots, x_n]$.

Any polynomial expression can be written using `Plus` and `Times` in conformity with the rules we have just stated, together with the minus-sign to denote the operation of taking the additive inverse. For example:

$$1 + x + x^2 = \texttt{Plus}[1, x, \texttt{Times}[x, x]];$$

$$(x+1)(y-2) = \texttt{Times}[\texttt{Plus}[x, 1], \texttt{Plus}[y, -2]];$$

$$x\,y + (1+x)(x\,y + (x+1)^2) = \\ \texttt{Plus}[\texttt{Times}[x,y],\texttt{Times}[\texttt{Plus}[1,x],\texttt{Plus}[\texttt{Times}[x,y],\texttt{Times}[\texttt{Plus}[x,1],\texttt{Plus}[x,1]]]]].$$

We can reveal the structure of a polynomial expression by writing it with `Times`s and `Plus`s and then deleting all the constant and variable symbols and minus signs and retaining only such commas and brackets as are needed to group occurrences of `Plus` and `Times`. For example, the three polynomials above yield `Plus[Times]`, `Times[Plus, Plus]` and

$$\texttt{Plus}[\texttt{Times},\texttt{Times}[\texttt{Plus},\texttt{Plus}[\texttt{Times},\texttt{Times}[\texttt{Plus},\texttt{Plus}]]]]].$$

Any `Times` (respectively, `Plus`) other than the head of the entire expression appears as an argument of some `Plus` (respectively, `Times`), which we say is *immediately above* it . We call an `Plus` or an `Times` a *terminus* if it does not have an `Times` or and `Plus` below it. From any terminus, we can read upwards to the head of the whole expression, or from the head we can read downward to a terminus. (Going *up* means going outside of brackets, going down means going inside.) If we read from the head to a terminus, we get a *branch*. For example, the branches in `Plus[Times, Times[Plus, Plus[Times, Times[Plus, Plus]]]]]` are `PlusTimes`, `PlusTimesPlus`, and `PlusTimesPlusTimes` and `PlusTimesPlusTimesPlus`. Clearly, every branch is an alternating sequence of `Plus`s and `Times`s. The `Times`-`Plus`-*complexity* of an expression is the number of `TimesPlus` s in the longest branch.

The distributive law transforms any expression of the form $\texttt{Times}[\texttt{Plus}[\cdots],\cdots,\texttt{Plus}[\cdots]]$ into an expression of the form $\texttt{Plus}[\texttt{Times}[***],\cdots,\texttt{Times}[***]]$. For example:

$$(u+v)(w+x)(y+z) = uwy + uwz + uxy + uxz + vwy + vwz + vxy + vxz.$$

Thus, the distributive law reduces the the `Times`-`Plus`-complexity of any expression with head `Times`. Repeated applications of the distributive law, therefore, ultimately result in an expression of the form $\texttt{Plus}[\texttt{Times},\ldots,\texttt{Times}]$ (or `Times` or `Plus`). The `Times`-`Plus`-complexity 0 of such an expression is zero.

Problems

1. This problem refers to expressions a)–f) above.
   a) Evaluate the expressions.
   b) Describe in words the computations that correspond to each.
   c) Draw the tree diagrams for c), d) and f).
2. How many different **numbers** can you form, using only addition and multiplication and using each the numbers from 1 to 5 at most once.
3. How many different **expressions** can you form, using only addition and multiplication and using each the numbers from 1 to 5 at most once.
4. Try Problems 2 and 3 with the numbers from 1 to 6. Try with the numbers from 1 to 7.
5. Tom says, "The distributive law tells you that any computation involving additions and multiplications—no matter how long and complex and no matter how many alterations between additions and multiplications—can be done by doing some multiplications first and then doing some additions." Is this correct?

Note. The tree diagrams were made using *Mathematica*. The input for c), for example, was

```
TreeForm[("1"+"2")"3"+"4")"5"]
```

It is necessary to put the numerals in quotation marks because without them, *Mathematica* evaluates the expression. The input

```
TreeForm[((1 + 2) 3 + 4) 5]
```

gives the output `65`. *Mathematica* applies some ordering rules before making the tree, so what it produces will not always have its leaves (the entries at the ends of the branches, at the bottom of the picture) in the order that you put them in.