The Expression Class

February 1, 2014

An expression Class is the core of a mathematical expression expressed as a tree. An expression tree could for example look like this:
This is the tree for the mathematical expression: $\frac{2^2}{3} \cdot b + \sin 5$

[.Addition [.Product [.Division [.Potens [.Number 2] [.Number 2]] [.Number 3]] [.Number b]] [.Sine [.Number 5]]]
In Python, this would be written as:

```
addition([product([division([potens([number(["2"]),number(["2"])]),
number(["3"])]),number(["b"])]),sine([number(["5"])])])
```

But don't worry; that's just the structure. If you want to convert $\frac{2^2}{3} \cdot b + \sin 5$ to the above, the function TextToCAS(str) will do just that. It's located in Data/textparser.py

Here's a table of all the expressionclasses, an example, their input, and special notes.

Class name	Examples	input	notes
addition	$a+b+c$	An array of **over 1** expression classes (addends)	
product	$a \cdot b \cdot c$	An array of **over 1** expression classes (factors)	
division	$\frac{a}{b}$	An array of **2** expression classes (numerator and denominator)	
potens	a^b	An array of **2** expression classes (root and exponent)	
number	a, 3, k_a	An array of **1 string**	This is the ”last stop” for any expression tree. It’s as deep as it goes
Unknown function	$f(x)$, $g(x,y)$,$P_3(v)$	A function string (”f”, ”g” and ”P_3” in the exmaples) and any **non-empty** array of expression classes (arguments)	This is just an unknown function, which can be simplified if the function is defined
sine	$\sin a$	An array of **1** expression class (arg)	
cosine	$\cos a$	An array of **1** expression class (arg)	
tangent	$\tan a$	An array of **1** expression class (arg)	
arcsine	$\arcsin a$	An array of **1** expression class (arg)	
arccosine	$\arccos a$	An array of **1** expression class (arg)	
arctangent	$\arctan a$	An array of **1** expression class (arg)	
natlogarithm	$\ln a$	An array of **1** expression class (arg)	
comlogarithm	$\log a$	An array of **1** expression class (arg)	$\log_{10}$
squareroot	$\sqrt{a}$	An array of **1** expression class (arg)	

Notice how every expression class has an array of other expression classes as the inputs. There are only 2 classes that has other inputs: The number class, which accepts an array of one string, and the unknownfunction class that accepts a functionsstring, and then an array of expressionclasses. And expressionclass must have the following methods:

- **type():** Returns the string of the name of the class, eg ”product”
- **tostring():** Returns the string of the expression, eg $2^3 \cdot b$
- **tolatex():** Returns the string of the expression in LaTeX, eg ”frac{2}{3}”

- **simplify(focus):** Returns a simplified expression of itself, or itself if no simplifications can be made. The focus is an optional number instance, which will make the simplifier simplify thinking you want to solve for that number. This method is defined for all expression classes in the SimplifyAll function in Entityclass.py
- **maxleveloftree():** Returns the maximum depth of the expression tree
- **evaluable(approx=False):** Returns a bool saying if the class instance could be simplified to a known number like 3 or 4.3 (not a number like k). If approx=False, then it will return True if the expression can be simplified to an integer, and if approx=True, it will return True if the expression can be simplified to a float
- **evalsimplify():** Evalsimplify will try to simplify the evaluable parts of itself, such that $2 \cdot 3 \cdot a$ becomes $6 \cdot a$. If it can't, it will return self
- **simplifyallparts(focus):** Returns a version of it self with all parts (addends or args, or (numerator and denominator) for example) simplified
- **contains(varstring):** Returns if the number of the varstring is in the expression, eg if x is in $2 \cdot x$ (this will return True)
- **approx():** Will just return evalsimplify(True). This way, it'll approximate the expression
- **__eq__(self,other):** Returns if one expression is equal to the other (they're not compared by .tostring())
- **compareinfo():** Returns a list consisting of self.type(), and the an array of whatever it got as an input, which means its addends, if the type is addition, factors if product etc etc
- **findvariables():** Returns a list of strings of the variables in the expression. "3*a*_m" would return ["a","_m"] (even though "_m" is a unit)
- **expand():** expands the expression. $(a+b)^2$ would return $a^2 + 2ab + b^2$. Is handled by the global function ExpandAll()
- **posforms(stringortex,approx=False):** Will return possible forms of the expression. Is handled by the global function posformsimplify in entityclass.py (basically it's a simplifier that simplifies for every focus)
- **printtree():** Prints the tree of the expression in an easily readable way. Used for debugging
- **makepossiblesubstitutions():** Calls the two dictionaries, and substitutes everything that is defined to something in the dictionary, eg if the expression is $2 \cdot a$ and $a := 3 \cdot \pi$, it will return $2 \cdot 3 \cdot \pi$
- **substitute(subthisexp,tothisexp):** returns the expression with subthisexp substituted for tothisexp
- **getsimplifyscore():** Returns an array of the maxleveloftree and then the length of the input array. Used to determine the best simplification

And that's it! The expression classes can have other methods, mainly simplifying methods, which i'll get to later.

An important rule about the expression classes, is that no method are allowed to change the values. It will only return another instance.