CAT Language Specification

Michael Faulkner

May 19, 2024

Introduction

CAT (Calculatron and Algebra Tutor) is a domain-specific language designed to assist individuals with mathematical calculations. More specifically, the language allows users to input mathematical expressions and operations to perform on the expressions. Running a program in this language will then produce a textual represenation of the ensuing calculation as well as a LaTeX file that can be compiled into a PDF that displays the calculation in a pretty manner.

While there exist computational tools for generalized mathematical expressions (for example Mathematica), such tools tend to focus only on getting to an answer. The purpose of CAT is to assist its users in understanding *how* to get the answer by outputting the intermediate steps of the calculation. This addresses another point of annoyance with tools like Mathematica that will sometimes output, seemingly arbitrary, restrictions on the parameters alongside its final expression. By working through the intermediate steps of the calculation, CAT should allow its users to understand why such restrictions were imposed and what the limits of the calculation would be if such assumptions were not made.

Design Principles

The primary goal of CAT is to produce easily understandable calculations. This goal corresponds with a prioritization of simpler techniques that are easier to follow over more powerful techniques that might be able to find more general answers but are less decipherable. This likely also means a tradeoff in terms of speed as it is necessary to work through the intermediate details of a calculation sufficiently to make them human readable.

Examples

Turns out teaching a computer math is hard... these tweaked examples reflect a shift in priorities I had. Basic symbolic manipulation of algebraic expressions has turned out to be a mucher rich (and way more challenging!) task than I initially thought it would be. Instead of calculus manipulations being a primary focus, I view them more as stretch goals (maybe I can do derivatives, but I don't think integrals are feasible), and I'm now primarily concerned with making the language display how you could manipulate an algebraic expression. CAT is capable of most core algebraic (as the below examples demonstrate); however, there are essentially limitless other simplification capabilities that could be added (see the 3rd example for a current limitation of CAT), so I am more concerned with making CAT produce nice output for its current capabilities than making CAT fully general.

1. The following example illustrates how each line of a CAT program is treated separately so long as there are no assignments. The first line would simplify to -1, and the second line would simplify to $2x$ (derivatives are not currently implemented; however, this example demonstrates how they would work).
   ```
   (x+1)(x-1) - x^2
   Differentiate x^2 + y^2 wrt x
   ```
   Sample output:
   ```
   Expanding:  (x + 1)(x + (-1)1) + (-1)(x^2)
   ==> (1 + x)(x + (-1)1) + (-1)(x^2)
   ==> (-1)(1)1 + (-1)1x + 1x + xx + (-1)(x^2)
   Simplifying:  (-1)(1)1 + (-1)1x + 1x + xx + (-1)(x^2)
   ```

```
==> (-1)(1)1 + (-1)1x + 1x + xx + (-1)(x^2)
==> -1 + (-1)1x + 1x + xx + (-1)(x^2)
==> -1 + (-1)x + 1x + xx + (-1)(x^2)
==> -1 + x + (-1)x + xx + (-1)(x^2)
==> -1 + x + (-1)x + (-1)(x^2) + x^2
==> -1 + 0 + 0
==> -1 + 0
==> -1

Differentiating with respect to x:  x^2 + y^2
==> 0 + 2x
Expanding:  0 + 2x
==> 0 + 2x
Simplifying:  0 + 2x
==> 0 + 2x
==> 2x
```

2. The following example illustrates how CAT can recognize the same expression in different forms and simplify them to a point where they can be combined. This also demonstrates CAT's robustness to unusual styles regarding whitespace and use of various symbols. This example should simplify to 0.

```
1+ (x+y) ( x - y )---(x) x + y^ ( 2 ) - z^((x+1)(x-1) - x^2 + y^(9 - 3^2))
```

The above calculation is quite tedious when every single step is written out, but I include it here to demonstrate the flexiblity of CAT. The initial expression (if you look past all the intentional garballing I introduced to challenge CAT) is:

$$1 + (x + y)(x - y) - - - xx + y^2 - z^{(x+1)(x-1)-x^2+y^{9-3^2}}$$

The LaTeX output produced by running CAT on the above expression is included here verbatim:

Simplifying: $1 + (x + y)(x + (-1)y) + (-1)((-1)((-1)(xx))) + y^2 + (-1)(z^{(x+1)(x+(-1)1)+(-1)(x^2)+y^{9+(-1)(3^2)}})$

$$1 + (-1)(-1)(-1)xx + (x + y)(x + (-1)y) + (-1)(z^{(1+x)(x+(-1)1)+(-1)(x^2)+y^{9+(-1)(3^2)}}) + y^2$$

$$1 + (-1)(-1)(-1)xx + xx + (-1)xy + (-1)yy + xy + (-1)(z^{(1+x)(x+(-1)1)+(-1)(x^2)+y^{9+(-1)(3^2)}}) + y^2$$

$$1 + (-1)(-1)(-1)xx + xx + (-1)xy + (-1)yy + xy + (-1)(z^{(-1)(1)1+(-1)1x+1x+xx+(-1)(x^2)+y^{9+(-1)(3^2)}}) + y^2$$

$$1 + (-1)(-1)(-1)xx + xx + (-1)xy + (-1)yy + xy + (-1)(z^{(-1)(1)1+(-1)1x+1x+xx+(-1)(x^2)+(y^9)(y^{(-1)(3^2)})}) + y^2$$

$$1+(-1)(-1)(-1)xx+xx+(-1)xy+(-1)yy+xy+(-1)(z^{(-1)(1)1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})})+y^2$$

$$1+(-1)(-1)(-1)xx+xx+(-1)xy+(-1)yy+xy+(-1)(z^{(-1)(1)1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})})+y^2$$

$$1 + xx + (-1)xy + (-1)yy + xy + (-1)(x^2) + (-1)(z^{(-1)(1)1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})}) + y^2$$

$$1 + (-1)xy + (-1)yy + xy + (-1)(x^2) + (-1)(z^{(-1)(1)1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})}) + x^2 + y^2$$

$$1+(-1)xy+xy+(-1)(x^2)+(-1)(y^2)+(-1)(z^{(-1)(1)1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})})+x^2+y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^{(-1)1x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^{(-1)x})(z^{1x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^{x})(z^{(-1)x})(z^{xx})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^{x})(z^{(-1)x})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)(3^2)})})(z^{x^2}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^x)(z^{(-1)x})(z^{(-1)(x^2)})(z^{(y^9)(y^{(-1)9})})(z^{x^2}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^x)(z^{(-1)x})(z^{(-1)(x^2)})(z^{(y^{-9})(y^9)})(z^{x^2}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^x)(z^{(-1)x})(z^{(-1)(x^2)})(z^{x^2})(z^{y^{-9+9}}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^x)(z^{(-1)x})(z^{(-1)(x^2)})(z^{x^2})(z^{y^0}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1})(z^1)(z^x)(z^{(-1)x})(z^{(-1)(x^2)})(z^{x^2}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1+1+x+(-1)x+(-1)(x^2)+x^2}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1+0+0+1}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^{-1+0+1}) + x^2 + y^2$$

$$1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + (-1)(z^0) + x^2 + y^2$$

$$1 + (-1)1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + x^2 + y^2$$

$$-1 + 1 + (-1)xy + xy + (-1)(x^2) + (-1)(y^2) + x^2 + y^2$$

$$-1 + 0 + 0 + 0 + 1$$

$$-1 + 0 + 1$$

$$0$$

3. This last example highlights CAT's current ability to factor expressions. CAT automatically expands expressions before simplifying them, so this example demonstrates how sometimes CAT will not recover the original expression. In a perfect world, this expression would simplify to `(x + y + z)^2`.
   ```
   (x + y + z)(x + y + z)
   ```
   Sample output:
   ```
   Expanding:  (x + y + z)(x + y + z)
   ==> (x + y + z)(x + y + z)
   ==> xx + xy + xy + yy + xz + xz + yz + yz + zz
   Simplifying:  xx + xy + xy + yy + xz + xz + yz + yz + zz
   ==> xx + xy + xy + yy + xz + xz + yz + yz + zz
   ==> xy + xy + yy + xz + xz + yz + yz + zz + x^2
   ==> xy + xy + xz + xz + yz + yz + zz + x^2 + y^2
   ==> xy + xy + xz + xz + yz + yz + x^2 + y^2 + z^2
   ==> 2xy + 2xz + 2yz + x^2 + y^2 + z^2
   ==> 2(xy + xz + yz) + x^2 + y^2 + z^2
   ==> 2(yz + x(y + z)) + x^2 + y^2 + z^2
   ```

Language Concepts

At it's core, CAT is a way of expressing and manipulating mathematical expressions. Each line of a CAT program constitutes an expression that will be evaluated. Evaluating an expression involves reducing it to the simplist form possible by applying operators and performing algebraic manipulations. The primitives of CAT are variables (i.e. x, y, z) and numbers (i.e. $1, -10, 3.141592$). These primitives can be combined with mathematical operations such as +, -, *, /, ^, (and maybe Differentiate) in order to form expressions.

Formal Syntax

```
<instruction>  ::=  <expr> ('\n' <expr>)*
<expr>         ::=  <ws>*<operation><ws>*
                 |  <ws>*<parens><ws>*
                 |  <ws>*<literal><ws>*
<operation>    ::=  <expr><op><expr>
                 |  <expr><ws>+<expr>
                 |  <expr><parens>
                 |  <parens><expr>
<op>           ::=  + | - | * | / | ^
<parens>       ::=  (<expr>)
<literal>      ::=  <number>
                 |  <variable>
<number>       ::=  <digits>
                 |  -<digits>
                 |  <digits>.<digits>
                 |  -<digits>.<digits>
                 |  .<digits>
                 |  -.<digits>
                 |  <digits>.
                 |  -<digits>.
<digits>       ::=  <d><digits>
                 |  <d>
<d>            ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<variable>     ::=  a | b | c |...| x | y | z
<ws>           ::=  Any non-newline whitespace character.
```

Semantics

Syntax	Abstract Syntax	Prec/Assoc	Meaning
`<number>`	Number of double	N/A	`number` is a primitive. I represent numbers using the F# double datatype.
`<variable>`	Variable of char	N/A	`variable` is a primitive. I represent variables using the F# char datatype. These represent symbolic variables that are used in mathematical expressions.
`<expr> ('\n' <expr>)*`	Sequence of Expression list	0 / left	`Sequence` consists of a list of expressions to be simplified. Every CAT program has one `Sequence` which consists of each line of the program. After evaluating, each expression within a `Sequence` is converted into a list of expressions that represent progressive simplifications of the original expression.