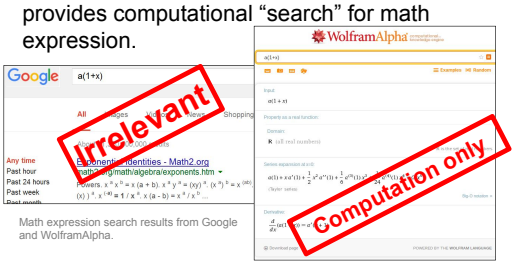
OPMES: A similarity search engine for mathematical content [1]

Wei Zhong and Hui Fang
University of Delaware

Motivations

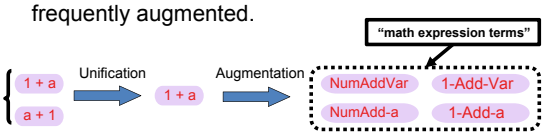
- Popular search engines are unable to search math by similarity. Even WolframAlpha only provides computational "search" for math expression.



Math expression search results from Google and WolframAlpha.

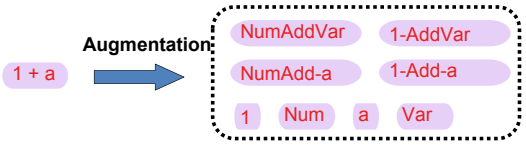
Motivations

- A lot research attentions are focusing on bringing existing general text model/tools (e.g. bag of words model, Apache Lucene) into math similarity search. These methods (text-based methods) inevitably requires unification process and large storage space as expressions are frequently augmented.



Motivations

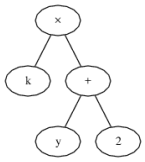
- To search expression by its subexpressions, text-based methods need even more space.



Our choice

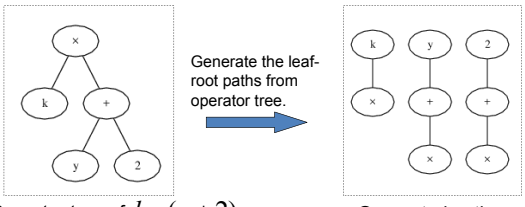
We choose a tree-based method to generate an intermediate operator tree so that we can extract structural information of math expression, at the same time avoid augmentation.

Operator tree of $k \times (y+2)$:



Intuition to use tree

An operator tree uniquely determines the leaf-root paths decomposed from the tree, no matter how the operands are ordered.

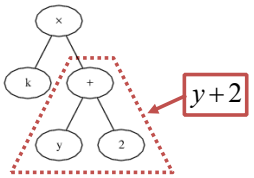


Operator tree of $k \times (y+2)$ Generated paths

Intuition to use tree

A subtree of an operator tree also represents a sub-expression:

Name this “subtree-subexpression property”

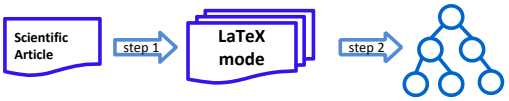


$k \times (y+2)$

Parsing LaTeX into tree

We implement a LALR parser to convert math formula (LaTeX input) into operator tree. (thus not dependent on *LaTeXML*):

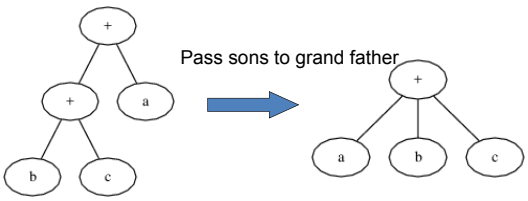
step 1: Crawl and extract math expressions in LaTeX



step 2: Convert LaTeX math expressions into operator tree using a LALR parser.

Parsing LaTeX into tree

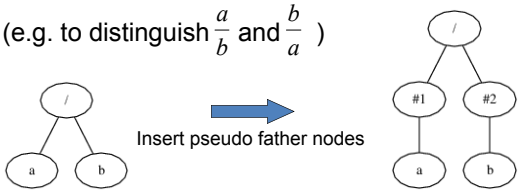
Case 1: If a commutative node has a father operator who is also commutative, the node will pass its children to its father and delete itself.
(Ensure subtree-subexpression property)



Parsing LaTeX into tree

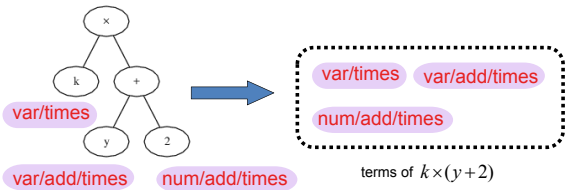
Case 2: When non-commutative operator is being constructed, insert different pseudo nodes on top of its children.

(e.g. to distinguish $\frac{a}{b}$ and $\frac{b}{a}$)



Leaf-root path as expression term

By using leaf-root paths as expression terms, we get only at most *O(L)* terms per expression, where *L* is the number of leaves from operator tree of that expression.

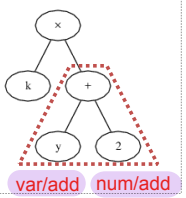


terms of $k \times (y+2)$

Leaf-root path as expression term

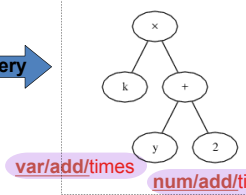
At the same time, we are able to search expression by its sub-expression, through matching the prefix of leaf-root paths.

$y+2$



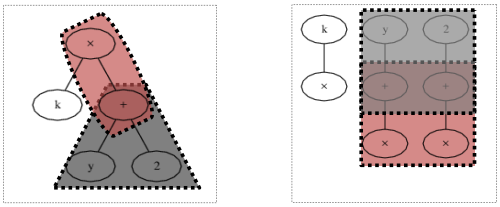
query

$k \times (y+2)$



Leaf-root path as expression term

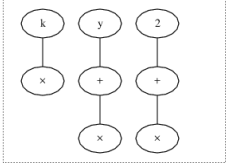
And leaf-root paths from the subtree must share some common nodes (red shadow) from the root of parent tree to the root of subtree.



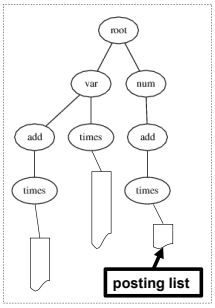
Index and search

As a result, we tokenize and insert leaf-root paths into a global persistent (on-disk) "prefix tree" as our index. Expression IDs (exprIDs) are appended to "posting list" under each inserted leaf-root paths.

Leaf-root paths of $k \times (y+2)$

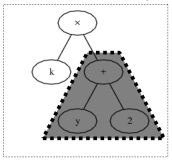


$k \times (y+2)$
Indexed

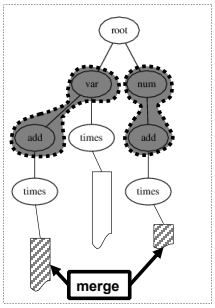


Index and search

We search simultaneously along the way of all leaf-root paths in the index tree, and merges the exprIDs from corresponding posting lists of query paths. If the deeper level nodes have a common node, we go simultaneously into that common node and repeat this process recursively. In this way we are also pruning indexes that are not common at the deeper level.



search an expression by its sub-expression.



Scoring

Every merged exprID is treated as an expression that is "structurally relevant" to query expression and is considered as a hit. Then "symbolic" evaluation are applied to structurally relevant expressions in order to score and rank results.

$e = mc^2$
Query expression

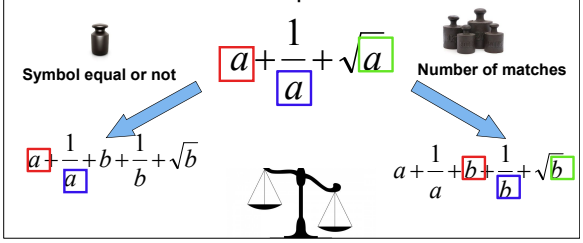
search

- $E = mc^2$
- $y = ax^2$

Expected ranking

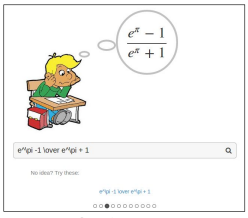
Scoring

Our proposed algorithm Mark-and-Cross [2] is used to score symbolic relevance degree. It also considers α- equivalence:

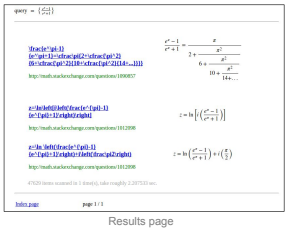


Demo

A demo page of our system is available at http://tkhost.github.io/opmes



Query input page



Results page

Demo

- Parser output (operator tree ASCII graph and leaf-root paths)
- Index tree structure
- A simple query-to-results explanation.

1. Clone source code from https://github.com/t-k-/opmes
2. After building (simply type ***make***) the project, run ***parser/parser.out*** to see parser output given an input LaTeX mode string.
3. Type ***make demo*** and view a demo index tree under directory ./col
4. Search a simple query by typing
   ./search/search.out -n -q '1/2 (n-1)!'

References

- [1] Wei Zhong, Hui Fang. A similarity search engine for mathematical content. http://tkhost.github.io/opmes/ecir2016.pdf, 2016.
- [2] Wei Zhong. A Novel Similarity-Search Method for Mathematical Content in LaTeX Markup and Its Implementation. http://tkhost.github.io/opmes/thesis-ref.pdf, 2015.