Math Search with Equivalence Detection Using Parse-tree Normalization

Abdou Youssef
Department of Computer Science
The George Washington University
Washington, DC 20052
Phone: +1(202)994.6569
ayoussef@gwu.edu

Mohammed Shatnawi
Department of Computer Science
The George Washington University
Washington, DC 20052
Phone: +1(571)274.9289
shatnawi@gwu.edu

ABSTRACT

Searching the Web for a mathematical expression is not an accurate process; the result of the search is unexpected most of the times. The inaccurate result is due to the nature of the mathematical expression search process, which is not based on clear and structured rules. In addition, the available techniques are not applicable to such expressions but they are designed and tailored to work with normal text together with different kind of documents (*e.g.* multimedia). For example, when you type y+x and hit Google search button, you are not sure if you can get most of the documents that have y+x and the ones that contain the variant of y+x (*e.g.* x+y). Searching for y+x gives some documents that have x and y (and maybe the character +) as separate characters but not x+y as a one term mathematical expression.

The purpose of this research is to explore some proposed normalization rules then to develop a general way that can be utilized to transform a user input expression into a normalized form. Accordingly, this process will create some kind of standardization between a mathematical expression as a search term and the contents of the database itself. Those rules at the beginning of this research are fixed based on the properties of mathematical operations (e.g. addition operation is associative and commutative, etc), this type of normalization between the mathematical expression and the searchable database assists in increasing the accuracy of searching in terms of evaluation measurements (Recall, Precision). This research will end up with a general way of transforming a user expression input into a normalized form that is necessary to enhance the searching process for a mathematical expression. The general way of transforming the expressions into their normalized forms is based on our proposed Grammar of Equivalence Rules (GER) subsystem, which contains some of predefined rules in which a system can chose from based on the user input itself. The system may apply only one rule, two, or any number of rules as appropriate to get the expression in its unique normalized form. The normalized form will be searched for against a searchable database.

Key Words: mathematical expression parser (MEP), equivalence detection and normalization (EDN), rules of equivalence, grammar of equivalence rules (GER).

1. Introduction

Finding needed information on the Web is not easy to achieve with a high degree of accuracy. Information retrieval systems have been designed to help users locate and retrieve their requests on the Web. Information retrieval systems are composed of some algorithms that try to make the search and retrieval of the requested information as accurate and fast as possible.

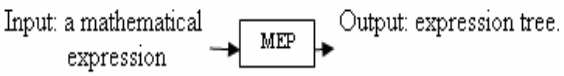
The kind of information that information retrieval systems deal with is comprised of text, images, audio, video and other multimedia objects. Among all of these, "the text aspect has been the only data type that lends itself to a full functional processing" [2]. Many algorithms that work together trying to refine text search have achieved a good level of maturity. For example, Google is a good example of such search engines. Google has achieved much more satisfying results than other search engines in terms of text-based search. Unfortunately those search engines did not achieve the same progress in terms of mathematical expression as a separate distinguished type of text

The major obstacle to math search in current text search systems is that those systems do not differentiate between a user query that contains a mathematical expression and any other query that contains text term. Therefore, they process mathematical expressions as other texts, regardless of its nature of being well-structured and having properties that make it different from other forms of text.

Here, in this context we will try to refine the text search process that is specialized in searching for mathematical expression. We will add more algorithms to the Information Retrieval System in order to make it suitable to do search for a mathematical expression as well as other forms of text.

2. Expression Parser

The first step of our work is to create a Mathematical Expression Parser (MEP). MEP is part of the software that takes a mathematical expression as input and creates its expression tree. The work process after creating the expression tree is based on the expression tree itself. Therefore, tree representation is used for equivalence detection and normalization.



Example 1: The expression y+x is translated into + using the MEP.

```
  +
 / \
y   x
```

3. Equivalence Detection and Normalization (EDN)

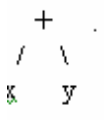
The equivalence detection and normalization is the most important part of our work. Indeed, it is the core of our research. The EDN aims to transform the expression tree that we have created earlier using MEP into a normalized tree. This tree is equivalent to the original tree but it is an agreed upon representation, based on some rules, to facilitate the search process. Therefore, the normalized tree should be the common form between the searchable database and the mathematical expression as a search term.

Using Google search engine to search for the expression x+y gives inaccurate result since it gives documents that have x, and y without the character + in between, or documents that have x/y/z but not x+y. The reason behind that is that Google uses techniques for matching and probabilities of occurrences of x and y. Google does not only search for the infix mathematical expression x+y. For example, Google retrieves the documents that have a high frequency of x (and/or y) occurrences. In spite of the above result of Google search, Google may work and retrieve some of the documents that have a high frequency of x, y and + occurrences. But still this result is not the one we are looking for; despite the

fact that those documents have x, y and + they may not appear in the required order (*i.e.* x, +, then y). For example, Google retrieves the documents that have xy+yx, which means x, y and + appear as part of other expressions but not a stand alone expression (*e.g.* x+y). In order to increase the accuracy of searching for a mathematical expression, we need techniques that work better with these searches.

Throughout this research paper we propose four fixed rules that can be applied to the above tree (example 1). Therefore, after applying them as needed, we shall be able to get the final normalized tree form. After that, the last normalized tree is used for comparison and matching during the search process.

Before getting to those rules, let us study a simple example so we can understand the whole process. For example, suppose the user query is y+x. Without equivalence (normalization) detection, a math search system will not retrieve documents containing x+y. The reason is that, typically y+x is stored as normal text in the searchable database. So, text matching does not work properly with this kind of mathematical terms. Alternatively, math expressions may be stored as parse tree structures. For example, x+y may be stored in the form of:



If the search term is transformed to its tree representation, which is  , then a straightforward tree-matching algorithm will not work for search.

Thus, the above form is not matched with the one stored in the database even though we have one equivalent to it already in the database, which is x+y. In this research paper, we will make both terms equivalent by applying some rules called the Rules of Equivalence. For the above example, we can apply the rule of reordering the tree elements; therefore, the tree of y+x is reordered alphabetically to be x+y since x comes alphabetically before y. Applying this rule results in a new tree representation which now matches the one in the searchable database. Thus far, we can start the interpretation of each rule with more details since the idea of our work has become more understandable.

4. Rules of Equivalence

In the first part of this research we will study a few fixed rules of equivalence. Then after finishing this part we will expand our work to be more generalized and more applicable to include most of mathematical expression search process.

Those rules have been implemented using Java.

4.1 Group Removal Rule.

A mathematical expression is grouped if it appears between left and right parentheses. For example, in the expression (d+c) +f^-a/3 (This example will be used throughout this section), the first + operators has given the highest priority of execution because of the parentheses (Grouping).

It is obvious that the above expression can be transformed to the following expression tree using MEP:

Figure 4.1.A

According to our software, the expression tree is drawn from the postfix notation of the above expression. Because the postfix notation has been used to draw the expression tree, there is no need for parentheses to represent the operations precedence (*i.e.* which one should be executed first). The postfix notation implicitly implies the operations precedence. Also because the ( ) node has only one child, therefore, the ( ) node can be removed. So the tree representation will look like the following tree:

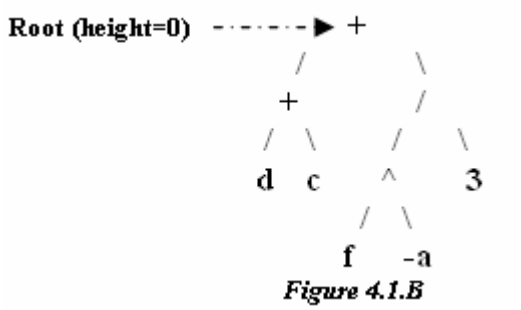


Figure 4.1.B

Even this rule is simple but it helps the normalization process and it achieves a little performance gain since it decreases the tree height as you can notice from the left sub tree of the root.

4.2 To the Negative Power Rule

Throughout this paper we assume the power sign is represented by ^ character. The previous example has "to the negative power" sub expression as part of it (*i.e.* f^-a). This part can be transformed to an equivalent expression by using the following mathematical rule:

- x^-y is equivalent to 1/x^y

Therefore, according to this rule, the previous expression, after applying the first and the second rules, should be transformed to d+c+1/f^a/3 (notice the grouping has been removed based on the first rule), which is transformed to the following expression tree:

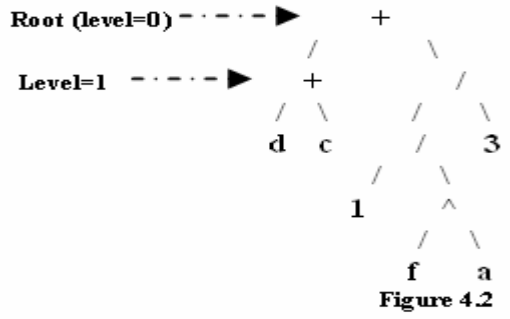


Figure 4.2

This rule is important in terms of normalization and equivalence as we will explore later in this research, even thought it adds a little burden by increasing the tree height.

4.3 Tree Height Compression

The time for some of the tree operations (*e.g.* find a depth of a tree, delete a node), is measured by the height of the tree. Therefore, shrinking the height of a tree and widening it somehow is good for such operations because of performance gain issues. In this section we will follow the same procedure of decreasing the height of the tree by applying the rule of Height Compression. This rule works as follows:

All the similar parent nodes that are descending from the same node are combined with lowest level parent node. Therefore, the leaves will be children of that common node given that the parent of each of those leaves will not change but their level will be changed after applying this rule. For example, figure 4.2 contains two division operators and two addition operators. Those operators are minimized to be only one addition and one division operators. To do so we should be able to maintain the parent of each node after deleting the extra operators. Such as, the parent for the node that has 1 as its data should be maintained as it is before and after the compression. Achieving this can be done by adding one more child for the division operation in first level, thus we have division with 3 children instead of only two children.

Also we shall allow for each node to have as many children as needed, because as levels are compressed more children are needed. All of the above can be illustrated more by applying this rule on our example in figure 4.2. Therefore, the tree now would look like the following tree:

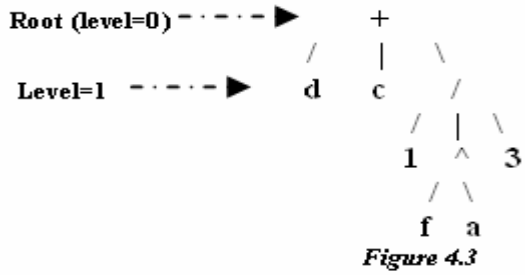


Figure 4.3

Notice that the height if the tree decreased from four to three. This has caused some performance gain in terms of any operations that come after this height compression.

4.4 Tree Reorder Rule

Taking advantage of the performance gain that we have achieved by applying the first

3 rules can be utilized better if the fourth rule is applied after them. For example, the third rule may decrease the height of the tree which improves the time for implementing this rule. Since accessing the leaves, in this rule, is required to reorder them (*i.e.* sort).
Sorting or reordering the leaves is done by following a user defined rule of reordering. For example, we proposed our defined rule, which is:

Numbers < Alphabetic (string, character) <Operations (, +) < Grouped Parenthesis*

Since we proposed the above rule, this does not mean that other users can not propose their own rule. But we have to apply the same proposed rule consistently on both the user query and the searchable database.
After applying this rule, the expression tree looks like the following:

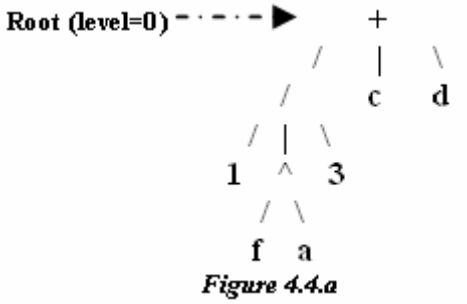


Figure 4.4.a

The above tree shows, in spite of a<f their order did not change because the operation between them is the power; this operation is neither commutative nor associative. On the other hand, the addition or multiplication operations are associative and commutative. You can notice the reordering of the characters c and d because c<d alphabetically. Note also the node that has the value 1 as its data comes first. The reason behind that is because the division operator is neither associative nor commutative.
The above four rules of normalization and equivalence can be summarized by the following diagram:

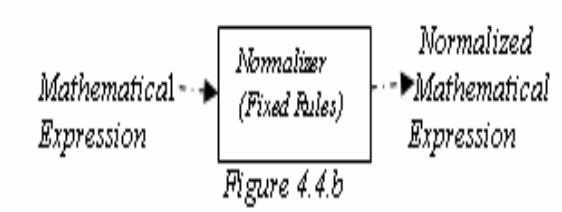


Figure 4.4.b

Our goal now is to explain how this normalization causes some performance gain in terms of helping in document retrieval and accuracy that we have talked about earlier. Information retrieval systems are evaluated by two main measures, which are Precision and Recall. Therefore, we will evaluate our system using these two measures. We should not forget that at the same time the normalization causes overhead because of the normalization computations.
This research paper focuses on the commutative and associative operations, such as addition and multiplication. Therefore, all of the above rules apply to those operations. But in the process of developing GER we will add as many rules of normalization as possible, trying to enhance the ability of our system and to generalize it to be a system that is capable of normalizing different kinds of mathematical constructs.

5. Measuring Performance and Overhead

.

5.1 Precision

Precision is defined as the ratio of the number of relevant hits to the total number of hits.

Precision= Number of relevant hits/ Total number of hits

The goal of our research is to increase precision, which is the ability to retrieve the most relevant items from the database, by increasing the number of the retrieved relevant items and/or of course decreasing the number of the total retrieved items.
Thus far, our work focused on the equivalence detection and normalization as the main tools in order to achieve a high degree of accuracy (*i.e.* high precision). By applying the above four normalization

rules we mapped many user inputs into one normalized form. That form is used then in the search process.
Using one normalized form for different inputs decreases the number of total retrieved items because the search process is focused on one normalized user input other than different ones.
Also this normalization process increases the number of the relevant hits because the searchable database contains the normalized forms of expressions, which results in increasing the number of true hits instead of false hits. For example, searching for the expression y+x+z may retrieve any expression that contains y+x+z or any equivalent form such as x+y+z

5.2 Recall

Recall is defined as the ratio of the number of relevant hits to the number of all relevant items in the searchable database.

Recal=Number of relevant hits/ Total number of relevant items in DB

Recall is used to measure the ability of the search in retrieving all the relevant items from the searchable database. But with the recall, measuring the number of all relevant items is not possible, except in controlled situations (where the "ground truth" is known). In this research we are concern with retrieving as many relevant items as possible and minimizing the number of irrelevant retrieved items.

5.3 Overhead Issues

If the search achieves 80% precision then the 20% of the user effort is overhead since the user spends time in reviewing irrelevant items (20% of the search result). If the user search results in 10 items and only 8 of them are related, then the user already has spent time to review the 2 unrelated items.
The overhead in the software that we have written is the time that the computer processor spends in performing any of the software different operations. The software aims to enhance the accuracy of the search process for a mathematical expression has some overhead issues to be discussed. Those overhead issues appeared in the different tree operations that we have implemented in our software. Generating the tree from a mathematical expression takes time and other tree operations after that take time as well. The four rules of normalization were applied in a tree form of the mathematical expression, therefore, whenever of applying a rule, visiting each node of the tree is required, which is taking some time. Also, comparing the two trees (*i.e.* the one in the database and the one for a user query) in order to search for an equivalent tree for any mathematical expression takes some time, which is an overhead by itself.

6. Comprehensive Performance Evaluation

Measuring the performance of any new-developed system is required to evaluate its reliability and to compare it with other existing systems. Therefore, we will carry out performance evaluation; measuring the improvements in precision and recall due to normalization.
The major problem in measuring the performance of math search systems is the lack of any math query benchmark because this area is relatively new. Therefore, in this context, NIST's Digital Library of Mathematical Functions (DLMF) will be used to evaluate our system's performance. I will use about 200 math queries that were developed by Prof. Youssef as a benchmark for evaluation [4]. The results of this preliminary research together with the results of our general normalization system (GER) will be available in later publications.

7. Conclusion

This research shows that we have achieved some progress in searching for a mathematical expression (*e.g.* y+x). Thus far, there is no such research that specialized with a mathematical expression only. In our research we focus more on

mathematical expression search process in terms of search engines and the Web search issues.
After applying the normalization and equivalence rules, the precision of our search will be increased. Ending the search process with a high precision is required in order to end up with an accurate result. Since we are transforming different equivalent mathematical expression into only a common form, this common form will be compared against the searchable database, which contains the normalized form of that expression as well. According to that, the comparison process will end up finding most of the items that has the common mathematical expression.
According to the above, our research is good in terms of enhancing the mathematical expression web search process. This way of enhancing is done by using fixed rules, but this research is part of an ongoing–research. The ultimate goal of this research is to create a general system that transform a user input, which is a mathematical expression, to a normalized unique form. The later is equivalent to the original user input. In order to transform the input expression into its normalized form the system applies a set of rules on the input expression.

8. Future Research.

In the preliminary research, the mathematical expression is transformed into a normalized expression based on some fixed rules (the above four mentioned rules). That normalized expression is compared against the searchable database trying to retrieve as many items that contain an equivalent expression to the normalized form as possible.
The new research will be focused on developing a general way in order to achieve the primary research goal. Instead of using fixed rules to normalize the input expression into a unique form, the new research will be using a Grammar of Equivalence Rules (GER) as pictured in the following figure:

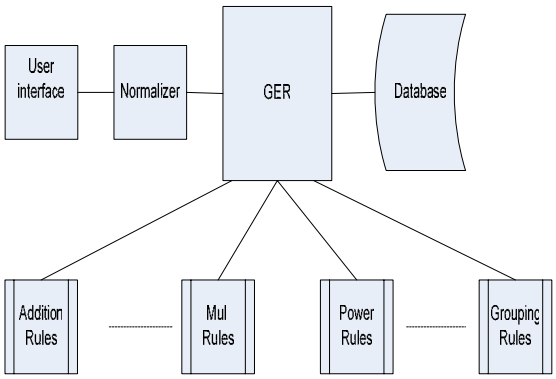


Figure 7.1

The above pictured system shows its main part, which is the Grammar of Equivalence Rules. Most of the work in the future research is related to this part. We will explain every single step of building that system. In the user's point of view the GER is hidden and the user will not notice its existence. What is matter for a user is just transform its input into an equivalent unique normalized form. Equivalent, unique, normalized and some other terms will be discussed and explained. Different issues associated with the GER will be discussed further in the future research.
When a user type a mathematical expression to search for on the web, the "normalizer" will communicate with the GER trying to find out applicable rules in the form of grammar that the "normalizer" can use to generate the normalized form for user input. The details of this new system together with its related issues will be published later on.

References

[1] Bruce Miller and A. Youssef, "Technical Aspects of the Digital Library of Mathematical Functions", Annals of Mathematics and Artificial Intelligence, Volume 38, pp. 121--136, 2003.

[2] Kowalski, Gerald J., Maybury, Mark T. "Information Storage and Retrieval Systems".

[3] K. F. Chan and D. Y. Yeung, "Mathematical Expression Recognition: A survey," Int'l Journal on Document Analysis and Recognition, Vol. 3, No. 1, 2000, pp. 3-15

[4] Abdou Youssef "Search of Mathematical Contents: Issues and Methods"

[5] http://www.hsl.creighton.edu/hsl/Searching/Recall-Precision.html

[6] http://www.cs.utexas.edu/users/mooney/ir-course/slides/Evaluation.ppt