Charles University, Prague
Faculty of Mathematics and Physics

MASTER THESIS

Vyhledávací stroj pro matematiku

Mathematical search engine

Jozef Mišutka

Department of Software Engineering

Supervisor: RNDr. Leo Galamboš, Ph.D.

Study Program: Computer Science

Prague 2007

Acknowledgements

I would like to thank my supervisor RNDr. Leo Galamboš, Ph.D. for his overall support. My big thanks belongs to my parents and to my sister for their support but especially to my father for his priceless remarks.

My thanks goes to Prof. Dr-Ing. Gerhard Weikum for his support during my stay at the Universität des Saarlandes in Germany. Furthermore, I would like to thank RNDr. Petr Pišoft, Ph.D. for his valuable comments and corrections of the draft of this master thesis.

I hereby declare that I wrote the thesis by myself and listed all used sources. I agree with making the thesis publicly available.

Prague, 7th August 2007 Jozef Mišutka

Abstrakt

Názov práce: Vyhledávací stroj pro matematiku
Autor: Jozef Mišutka
Katedra: Katedra softwarového inženýrství
Vedúci diplomovej práce: RNDr. Leo Galamboš, Ph.D.
E-mail vedúceho: leo.galambos@mff.cuni.cz
Abstrakt:

Vyhľadávacie stroje (napr. Google) vládnu obsahu dnešného WWW. Sú neoddeliteľnou súčasťou každodenného prístupu k informáciám. Veda, ktorá sa zaoberá práve vyhľadávacími strojmi, sa sústreďuje na vyhľadávanie prirodzených častí jazyka - slov. V posledných rokoch sa rozšírila pôsobnosť vyhľadávačov aj na iné oblasti. Veľkosť matematických dát na WWW enormne narástla. Dôležitosť matematických vyhľadávacích strojov je evidentná. Aj napriek tomuto vývoju, sa oblasť vedy skúmajúca matematické vyhľadávanie donedávna zanedbávala.

Napriek tomu, že aktívny výskum napreduje vo vývoji, zatiaľ bolo prezentovaných len niekoľko málo výsledkov. Hlavným cieľom tejto práce je vyplniť túto medzeru. Nový matematický vyhľadávací stroj bol navrhnutý so zameraním sa na použiteľnosť. Jediný vyhľadávací stroj schopný efektívne indexovať WWW - čisto textový (fulltextový) vyhľadávací stroj - bol použitý ako základ a vypracovaný návrh ako jeho rozšírenie. Toto umožnuje využívať všetky výhody fulltextového vyhľadávacieho stroja.

Väčšina matematických dokumentov neobsahuje informáciu o sémantike. Riešenie tohoto problému je základným cieľom tejto diplomovej práce. Testovanie ukázalo, že navrhovaný vyhľadávací stroj má viacero výhod. Najdôležitejšou je použitelnosť pre veľkú množinu matematických dokumentov s minimálnym sémantickým popisom, ako je WWW.

Kľúčové slová: *matematické vyľadávanie, matematický vyhľadávací stroj*

Abstract

Title: Mathematical search engine
Author: Jozef Mišutka
Department: Department of Software Engineering
Supervisor: RNDr. Leo Galamboš, Ph.D.
Supervisor's e-mail address: leo.galambos@mff.cuni.cz
Abstract:

The WWW is dominated by search engines such as Google. They are inseparable part of everyday search for information. Theoretical research field interested in searching, the information retrieval, focuses mainly on the natural language constructs - words. During the last years the field has been extended to other searchable content as well. The world of mathematical knowledge on the WWW has grown enormously. The importance of a general mathematical search engine is clear. However, this research field had been abandoned until very recently.

Despite the fact that an active ongoing research is in progress, few practical results have been presented. The main goal of this thesis is to fill this gap. A new mathematical search engine was proposed with the focus on applicability. As the only capable search engine of indexing WWW effectively is the full text search engine it was used as the basis. The mathematical extension was designed as an extension which allows it to exploit and use all the advantages of the full text search engine. Most of the mathematical documents do not contain semantic information. The solution to this problem was one of the main goals of this thesis.

The extensive evaluation showed that the proposed search engine has many advantages. The most important one is the usability over a large collection of semantically poor mathematical documents such as WWW.

Keywords: *scientific search engine, mathematical search engine, formula search*

Contents

1 Introduction

When internet was recognised as a very powerful tool it quickly began to grow. Very soon many problems arise. One of the most important problems was not the absence of information itself, but where and how to find it. Many different search technologies have been created to solve these problems. The first one was Archie providing only file name lookup, later Veronica and Jughead which were the first to index plain text. Both search technologies were using Gopher to acquire the text. It was simple but sufficient. And then the WWW came.

With the new concept of hypertext importance and need for new search technologies rose again. Among the first were web portals (also called web directories) and very primitive full text search. Both of them are still used. Directories are more likely to be used in local searches (e.g. http://centrum.cz). One global directory would be too immense and chaotic. Full text search can be used with any query (e.g. http://google.com, http://live.com) but the quality of results can vary. Results are sorted according to a ranking algorithm.

It could seem that these technologies are old but they are merely dated back to 1998 when Google came. The late development is very likely one of the key reasons why search engines are still simple. With expansion of the WWW came diversity. Crawling, indexing and searching became a complex problem. But still many document formats are not indexed or indexed only partially.

There are many specialised search engines which index only selected file formats like pictures, videos, music, code, maps, books. Each of them is based on a full text search engine with special recognition and indexing algorithms that extract keywords and assign them to files (http://video.yahoo.com, http://www.skreemr.com, http://codesearch.google.com/). However, there is no search engine indexing scientific documents on the WWW other than full text and it is only indexing simple text it can extract which is clearly not satisfactory.

The intense research activities in the field of scientific searching [6, 19, 19] clearly show the importance of scientific search engine. It can be used either to find scientific documents containing specific formulae without knowing proper names or to find scientific documents with similar formulae for example to prove the uniqueness of a theory. Moreover it can be used to simply classify documents and thus enabling better categorization. It can also be used to connect scientific research fields because many research groups from various scientific fields are analysing the same theory but from different angles without knowing about themselves. Another usage is to improve the performance of today's automatic theorem provers or the search engine can be used in proof planning systems for example as a proof assistant by collecting similar proofs. Another important usage example is in learning assistance where searches for proofs of various mathematical equations or formulae are common. Very useful can be searching for techniques used in similar proofs. All these applications with so many heterogeneous problems makes scientific search engine a very challenging research field.

The nonexistence of commonly used scientific search engine is caused by the number of problems which must be solved. They range from practical issues through document format problems to mathematical problems. Another important problem is that it must satisfy many different types of questions as can be see from the various applications. Although currently available text search engines can be used on these documents too, they are deficient in almost all cases. Except the obvious problems resulting from the mathematical aspect, indexing mathematical notations with science not aware search engine would have strange results. Ambiguous searches like "sin" or "a" would return documents containing sine function and the English noun sin or documents containing variable a and indefinite article or moreover the "a" is filtered by a stop word[1] filter. Another problem is that expressing scientific notations in graphical front-end of a text search engine is impossible.

Every search engine must operate over a set of data. Full text search engines normally operate over simple texts which are extracted from various formats. There are many tools which can fairly successful extract simple text from most of the document formats. But the scientific search engine must not only identify scientific text but also reconstruct it to its original or equivalent notation. The first task is a classification problem [10] where the text is classified into scientific text and simple text. The second one can be described as a recognition problem. Both these problems are important parts of computer science.

There are many possibilities how to publish scientific text each of them resulting either in a picture like format or a highly-structured notation. Several organisations tried to introduce a standard for mathematical representation in the last years. Without a standard the development of a serviceable scientific search engine is almost impossible. The most promising and used document format containing support for mathematical semantic information is `MathML`.

Information retrieval has been one of the main research areas in computer science. One part of the information retrieval is also searching and generating additional information called metadata (including context data) from the document. Search engines can index text documents with little or completely without any document's metadata and still be very valuable as today's most favourite text search engines prove. The basic impression is that the scientific search engines must heavily depend on metadata as many results are valid only in special theories or under special circumstances. Many papers [3, 5, 18] suggest that approach to scientific searching leaving out the semantic information is not sufficient. This statement is on one hand clearly a fact

[1]A stop word is a commonly used word (such as "a" or "the") that a search engine has been programmed to ignore.

from the mathematical point of view but on the other hand rather ambiguous and vague as the full text search engines already proved that the lack of document's metadata and context data does not mean less valuable results. Besides to fully use the metadata the user would have to include metadata about his query too which would be very limiting.

Supported features and graphical interface of a full text search engine has been standardized over the last few years. But there is no such standard in scientific search. There are many experimental designs and mostly the search engines offer more ways to express the query. It is partly because no real scientific search engine exists, partly because many people have different expectations from it and partly because every existing scientific search engine has still limited scientific awareness [37, 23, 34, 40, 38, 33, 39]. One feature is for granted when a search engine is to be considered truly scientific and that is the support for searching scientific notations and symbols. Mathematical formulae, chemical formulae, dental formulae, vertebral formulae and others are included in the term scientific formulae. But because these notations can be described by mathematical formulae and it is very likely that mathematical formulae are the most common in internet the term scientific search engine and mathematical search engine can be used interchangeably.

Last but not least problem connected with scientific searching is the fact that most of the documents are author's intellectual property and the access to them is restricted to paying subscribers only. Papers are usually published in interactive databases [41, 32] and the access is granted only to paying subscribers. This is the reason why published papers are not indexed by common search engines. However, the abstracts are normally made available and indexed.

The first workshop about scientific searching was held in 2004. The main conclusion was the need for searching, especially in scientific documents. `XML` and `MathML` were considered as very important for publishing scientific documents. Many research groups are addressing issues connected with scientific searching [27, 28, 29, 30, 31].

1.1 Goals of the Thesis

Scientific searching is still very academic. It faces many problems from complexity of mathematics through the lack of standards in electronic presentation to the definition of what a scientific search engine should look like and which features should it contain.

To create a scientific search engine these new problems must be solved: 1) the extraction of mathematical content from mathematical documents, 2) classifying simple text and scientific notation, 3) indexing of scientific text and the design of the ranking algorithm, 4) design of the query language. The approach presented in this paper tries to exploit the current state of art of search engines and to present a practical approach to scientific searching. The goal is to propose a new approach addressing almost all of the fundamental problems above with the accent on practical usability.

Extracting and classifying content from mathematical documents include both converting the document to supported format and parsing the format to at least syntactically equivalent form. The first issue is very complex and the results can be ambiguous. Both issues are analysed in [11].

The successful addressing of the third problem also includes brief analyse of the state of art of available scientific aware search engines and reasons for their advantages, disadvantages and reasons for their failure to become a general practical solution. The main goal of this thesis except the general proposal of a search engine is the design of a novel indexing technique. The design of general ranking algorithm is a very complicated and complex research field and thus this issue will not be analysed in depth but rather proposals and remarks will be made based on the results.

Another important goal is to evaluate the proposed solution and decide whether it is another failure or success with the potential to become a practical solution.

This is a summary of issues addressed in this paper:

1. problems connected with the scientific searching, state of art of existing solutions and the reasons why they are not suitable for large collections of documents
2. design of scientific search engine
3. proposal of scientific notation indexing technique and ranking algorithm
4. proposal of scientific query language
5. prototype implementation
6. summary based on the evaluation results

1.2 Structure of the Text

The definition of a scientific search engine with an architecture overview is described in Chapter 2. The theoretical background of indexing with solutions to presented issues is in Chapter 3. The searching phase and the description of a searching query language is in Chapter 4. The prototype implementation is described in Chapter 5. Practical results and statistics are summarised in Chapter 6.

1.3 Related previous work

The area of full text search engines is developed very well. There are also many different approaches in scientific searching like theorem provers and proof checkers [14, 23], engines using statistical models (similarity, frequency of occurrences), substitution and unification [37, 38, 39] and specialized databases [34, 33, 40]. The only part really connected to this work are the statistical models more precisely similarity models.

At the time of analysing the state of art of the available scientific search engines, developing the theory and proposals of new search engine (fall of 2006) the only work publicly available connected to approach presented in this paper was the [22].

It is a very brief introduction originated in 2002 and from that time till the end of analysis no valuable work known to the author had been made available. Other mathematical aware search engines have been still either in research phase or in development. *MathDex* search engine beta version gone public in the first half of 2007. *LeActiveMath* is a not publicly accessible project of web-based learning environment called *ActiveMath* running till 2007. The time line of *MathWebSearch* is not known but the earliest paper appeared at 2006 and the publicly available implementation has been very likely not put online before 2007.

Because of such an extensive and rapid development of this field mainly presented at Mathematical Knowledge Management Conference 2007 hosted from the 27th to the 30th of June 2007 the newest results had to be analysed and incorporated into this work. New research results are still being published at conferences and the latest results used in this paper date to July 2007.

Therefore it can be claimed that no previous research known to the author but [22] has addressed the issues of using full text search engine as the base search engine for scientific search engine and the issue of designing indexing and querying algorithms for it.

2 Mathematical Search Engine

The importance of mathematical search engine is clear. There are few novel solutions developed specially for this purpose but none of them seems to be practically applicable for large collection of real documents. Real documents usually do not contain any semantic information only presentation. To fill the gap a new solution is presented which works with documents not containing semantic information, but uses it when available, and which tries to exploit the current state of search engines with the stress on applicability. Every text search engine can be easily extended to adopt it because it is designed as an extension.

2.1 Full text search engine overview

Search engine is an information retrieval system aimed to help finding information. It operates over information stored electronically on local computers or networks. There are few different types of search engines. The most used search engines operate over the WWW and are called web search engines (http://www.google.com, http://www.yahoo.com, http://www.baidu.com). In this paper the general term search engine will refer to web search engine. Each search engine should meet several basic requirements to be successful. It must operate over all information available and integrate content changes as early as possible, return all relevant documents sorted by importance. Search engines must face many challenges and several interesting ones can be found in [2].

Each search engine has three fundamental phases either directly separable or mixed together: 1) crawling, 2) indexing, 3) searching. The brief description of each phase follows.

2.1.1 Crawling

Search engine receives documents for indexing mostly from its own program automatically browsing the WWW called crawler. Crawler starts by retrieving one of its starting addresses and then recursively all addresses to which the starting address links. Web search engines usually retrieve only sites from the surface web[1].

[1]Very simple definition defines it as a part of the WWW that is available to the general public without human interaction e.g. passwords, user input [4].

2.1.2 Indexing

In this phase the search engine takes a document, parses it to tokens, classifies its parts, collects other important information, ranks the tokens and finally stores them with other complementary information. This results in database update. Many special techniques are used to reduce and pack information stored in the index database. The information is usually not independent and a database update can result in a cascade of updates. Full text search engines use static databases - this means that in the searching phase the information from the index database is used as it is stored.

This part is considered the most important as the other parts are more or less standardized for all search engine types. But it is for granted that it is the most difficult part which joins many interdisciplinary research fields.

2.1.3 Searching

The only user visible part of a search engine is a graphical interface called search front-end. The trend is to have the search front-end as simple as possible. Inputted search term is usually encoded to the url, sent to a server and then the result page is retrieved showing relevant hits sorted by importance according to the search engine. Basically, forming a query to get relevant answers to more complex questions requires a very good imagination and practice. In the last days users of the most used search engines face a new challenge. They must win over the hard-coded algorithms deciding what user really wants in contrast to the query she inputted.

A search query language can be defined as words and commands sent to the search engine. The specific user input is called a search term. The search term must be parsed and then the index database is queried. The query language can contain commands like *OR*, *AND*, *""*. All favourite search engines support only few operators from all the possibilities they could. Still the statistics show that only very few of them are really used and the most common user does not use them at all (AOL Research).

The most usual characteristics of today's search engines are:

- smallest searchable unit is one word
- ranking algorithms are based on quality of document, backlinks[2] and semantic structure
- context used only during the indexing phase and it is difficult to utilize it afterwards
- query language supports publicly only simple boolean logic but internally more powerful operators (e.g. near operator[3]) are available
- possibility to limit search terms to a specific document type or document field (e.g. title, body)

[2]Backlinks are all incoming links to a web page.

[3]Proximity operators allow you to locate one word within a certain distance of another.

2.2 New challenges

The difference between simple text search and mathematical search is that it has to answer completely different types of questions. Whereas the text search engine generally answers the question which documents contain a very well defined text string or which documents are relevant to a text string the number of questions user would want a scientific search to ask is high. For example user would like to get all documents containing proofs of a theorem, new trends and knowledge from a research field, all research fields containing a specified theorem or formula, mathematically similar or equal formulae, numerical values of formulae etc.

With standard search engines in mind these new challenges have been identified: 1) enabling to search for scientific notations, formulae etc., 2) support for raw text queries, 3) search for mathematical notation should be limited to mathematical content (e.g. "sin" problem), 4) providing query language on one hand powerful enough to handle all mathematical notations but on the other hand simple to be usable by a common user, 5) support for basic mathematical operations to enable other forms of matching except exact hits e.g. similarity matching, pattern matching, 6) results should be ordered according to a ranking algorithm based also on the mathematical content.

As mentioned in the previous section the creation of a scientific search engine includes these main new problems which must be solved:

1. extract content from mathematical documents
2. distinguish between simple text and scientific notation
3. index scientific text and design ranking functions
4. design query language supporting mathematical notation

Solutions to these problems in the new proposed technique are addressed in the next section. To better understand the design decisions made in the proposal a brief analyse of available mathematical aware search engines and of new techniques that could be used for mathematical searching follows. Each description tries to briefly describe the technique used and main characteristics. Advantages and disadvantages of the mathematical approach are summarized at the end. The main goal of this thesis except the general proposal of a search engine is the design of a novel indexing technique. The design of general ranking function is a very complicated and complex research field and thus this issue will not be analysed in depth but rather proposals and remarks will be made based on the results.

2.3 Available mathematical search engines

There is a plenty of possibilities how to categorise mathematical search engines. Each possibility has a different point of view. The categories identified here definitely do not try to be exhaustive but rather present the most successful techniques used in the second and third phase of the search engine:

1. strictly mathematical model - the most precise solution when looking for mathematically equal notations. The problem is that it would have to contain millions of mathematical models to be suitable for the vague ambiguous world of internet and furthermore it would have to classify the models correctly.
2. substitution model - best results expected for searching aiming at similar presentation of formulae. Tries to unify two theorems by recursive substitutional rules, in the simplest form does not use mathematical axioms (like distributivity, associativity, commutativity). More practical for searching than the model above at the cost of losing information.
3. similarity model - best for interdisciplinary searches, searches with limited information. Even more practical than both models described above but at the cost of losing the most information.

Here is a list of techniques which could be used for scientific searching, available mathematical-aware search engines and also other search engines connected which scientific searching. Other techniques which have been analysed for possible use but found unsuitable like hashing and image matching are not included.

The three real solutions (*LeActiveMath*, *MathDex*, *MathWebSearch*) described below have been added after the MKM conference at the end of June, 2007. The analyse is based only on research papers and conference papers publicly available because the author of this paper has not attended the conferences. All questions sent to the authors of the mathematical search engines about the design or evaluation had been left unanswered but one. Therefore the analyse can be imprecise at some points.

2.3.1 Citeseer like

Citeseer search engine [34] does not use any of the model mentioned above because it is not a mathematically aware search engine. However, it is considered as a scientific search engine by many researchers. The main reason for its success is its specialization in searching through scientific documents with the paradigm of ranking based on backlinks introduced by Google.

Every well formatted scientific document contains at least bibliography section. This engine tries to exploit it and splits a document into fields significant for science documents [34]. The entries in the bibliography are considered as links to other documents. However, in many cases the lack of precise information about the author or title makes this search impossible.

2.3.2 Directories

The author does consider web directory to be a normal search engine. It must perform all search engine phases mentioned above and the user query is the specification of the directory entry. Science digital libraries [33] or databases [41, 32] exist all over the world. This search engine does not index document content but classifies documents into categories and indexes only these categories.

The success and application of a directory as a mathematical search engine is questionable. The advantages are that it is easily applicable for the WWW. It can be also used to obtain new results from a specified research field. The key concept used here is to assist the user in classifying the documents without the need to read it.

This technique can be easily deployed together with a search engine to improve the presentation and information about the results.

2.3.3 Xml searching

Scientific documents contain highly structured notations and therefore are a very good candidate for structured document format. In fact, the standard mathematical document formats (`MathML`, `OpenMath`) are `XML` based and therefore suitable for `XML` search engines [25]. There has been much development in this field because `XML` became world wide used with many different applications. There are many different approaches to the `XML` searching. Describing all of them is beyond the scope of this paper. To take the advantage of the structural approach, at least some information about the path to each attribute, element and value must be stored.

The only known research on this topic is in [21]. This is probably due to the fact that effective `XML` searching is still more an academic research field.

The basic `XML` processing query languages are XQuery and XPath. Both are very powerful but there has been no implementation usable for large collection of documents with these features: 1) complex transformations - needed for mathematical notations, 2) full text support. Another characteristic is that most of practical implementations do not support `XML` specification diversity but rely on the fact that the input data are based on single DTD or scheme.

The research is still at the beginning [26]. For instance a document is parsed to entities which are the smallest units of information. The key concept would be to construct the mathematical awareness at higher abstraction level than on raw formula string and that is at the level of the parsed entities. Because information about complete entity paths must be stored it is a natural process to retrieve the basic context information after the indexing phase. However, the process of searching over structures containing much information is very complex and usually only very limited queries are available.

2.3.4 Theorem provers and proof checkers

Computer programs specialised in proving of mathematical theorems are called *theorem provers* (E, Vampire, Isabelle, etc.) and in proof checking are called *proof checkers* (Mizar etc.). Both are important parts of automated reasoning. They operate over specified logic - set of axioms and transformation rules. The problem of deciding the validity of a theorem or a proof varies from trivial to impossible depending on the logic used (propositional, first order). For the common case the problem is decidable but NP-complete and hence only exponential time algorithms are believed to exist. They have special applications like software verification.

A theorem prover can be considered a mathematical-aware search engine as they search for theorems in a mathematical structure. They use the strictly mathematical model. Because proofs are constructed usually by using rewriting techniques it can not be used over a larger logic at present time. One characteristic is that this approach can not return results that "almost" match and that the query representing a theorem must contain all the semantic information about the mathematical structure where it should hold.

2.3.5 Semantic web

Semantic web is a relatively new, blooming research field in which information is expressed not only in natural language but also in a standardised way. The standard allows automated processing of such metadata. Current WWW is far from semantic but there are many sites already supporting it. Semantic metadata does not only describe objects more properly making the classification of documents and the content more precise, but also the relations between objects which current full text search engine can not use. Even though it is a very interesting and promising way to share scientific knowledge it is not in the near future to rely on the data to be available. Semantic searching can be very vaguely described as a combination of `XML` searching and reasoning about the search results. Theorem proving is a part of automated reasoning.

The practical solutions are still very feeble and much research must be done in order to decide whether this approach is suitable for mathematical searching. However, the main characteristics would be: 1) very precise classification of documents and notations (by assuming the document content is from a credible source), 2) support for the strict mathematical model, 3) very precise searches, 4) it can be seen as the natural way for automated reasoning (e.g. theorem provers) to access the WWW.

But even if the documents are described with reasonable metadata it is still limited by the knowledge of the author thus for example preventing completely new relations to be uncovered or similarity searches. The search would be processed on higher abstraction level like in `XML` searching. Another issue is the amount of overhead with which the semantic search engine would have to work and of course the computational complexity.

2.3.6 Encyclopedia of Integer Sequences

Encyclopedia of Integer Sequences is a very specialised mathematical database with its own search engine. As the name tells it stores integer sequences with additional information like where it can be found and more. Currently the database contains around 130 000 integer sequences. It is not truly mathematical aware search engine but proved very helpful in many ways including connecting different disciplinary research areas and their results.

2.3.7 LeActiveMath

`ActiveMath` is an intelligent web-based learning mathematical environment. The semantic content of mathematical documents is encoded in `OMDoc`[4]. The main goal is to present a learner personalised content based on her previous work, actual knowledge etc. Currently, the system does not provide public content and only subscribers or members can use it.

Obviously, the system needs a search engine. The requirements for this search engine are: 1) very easy to use allowing querying for text, metadata, mathematical formulae, 2) reasonably tolerant, 3) allow explorative learning. The `OMDoc` metadata in this project can be used to split the content to units called items: theorems, exercises, proofs, definitions etc. They can be addressed by unique identifiers and relations between them are also supported.

The indexing phase converts the `OMDoc` formulae into special text tokens with depth information which is then indexed. Although the ranking algorithm is very simple it can be still effective.

This specialized mathematical-aware searching engine is suitable for learners of mathematical content which do simple searches. The usability for more complex searches is questionable. When the depth information of subformulae is included in the indexed string the searching phase must convert the input formula to a representation including depth level too. The general search can not be used on searches for subformulae because we do not know at which level the subformula occurs. The solution to this is trivial but the problem gets more complex with increasing depth of the searched subformula. Another characteristic is that it does not use the semantics very much for the formulae itself which results in the fail when searching for commutatively equal subformulae (searching for $f(A, B) = f(B, A)$ requesting that the terms A and B are matched consistently is impossible). From the description of indexing phase follows that it does not support mathematical equality (commutativity - searching for $a + b$ should also return $b + a$) but this statement could not be proved.

The indexing phase fully depends on special format always including semantic information and metadata. The usability for real data is therefore very likely to be much smaller. However the use of special `OMDoc` format can boost the relevance of documents thus making this search engine very helpful in specialised environments. The important feature is that searching does not require precise definitions of the mathematical terms because it was designed for common users (students, teachers, researchers) which usually do not posses the knowledge.

[4]It is an extension of `MathML`, see http://www.omdoc.org/ for more details.

2.3.8 MathDex

`MathDex`[5] beta went public in the first quarter of 2007. It is one of the real solutions to mathematical searching aiming for these main features: 1) support for poor markup, 2) accepting many mathematical encodings, 3) allows searching on both mathematical notation and text, 4) attempts to match user text search expectations rather than strictly following the query. Between the first phase and the second phase all retrieved documents are converted to `XHTML`+`MathML`. The results are sorted according to the structural and syntactic similarity with the search term. It falls into the statistical model because matched formulae are not necessarily mathematically equal and the ranking is based on subformulae of a formula.

When analysing strings during the index phase not only simple word tokens are indexed but also the information about subformulae frequencies is collected. The information about frequencies is used to enhance the ranking of documents. This technique is also used when indexing mathematical formulae. Assuming we have all the frequencies of meaningful subformulae (x^2, $cos(x)$ or $sin(x)$ ...) we can use them to increment the ranking of complex formulae. To remedy some important problems with this algorithm fields have been introduced. It is the same principle as with fields in normal documents (title, body, etc). The mathematical formula is divided into numerators, superscripts, rows etc. The input formula is parsed, appropriate fields identified and the query is rewritten to match the subterms in the selected fields.

The main feature is the support of both mathematical and plain text search. Others include the rich support of input formats. The graphical user interface does not support other input possibility than graphical. The indexing algorithm uses only a special part of the available semantics and frequencies of formulae occurrence. Therefore pages with many formulae and notations (like summaries) will probably rank really high thus devaluing the quality of the results. Searching for $f(A, B) = f(B, A)$ requesting that the terms A and B are matched consistently is impossible.

2.3.9 MathWebSearch

The last representative of the small group of real mathematically aware search engines is MathWebSearch[6]. It is assumed that the first version was publicly available around the end of 2006.

It looks at formulae only from the semantic point of view which can be both an advantage and a disadvantage. The main goal is on one hand to distinguish between syntactically same notations but semantically different ($\int f(x)dx$ can mean Riemann but also Lebesgue integral) and on the other hand to identify and group syntactically different but semantically equal (${}_nC^k$, C^n_k, $\binom{n}{k}$, $\frac{n!}{k!n!(n-k)!}$).

Substitutional tree indexing technique is used to index mathematical notations. The result is a tree-like structure with nodes containing substitutions of its parents.

[5] http://www.MathDex.com:8080/MathDex/search
[6] http://search.mathweb.org/

The formula is constructed from a root node by applying one or more substitutions. It can be placed to the substitutional classification category.

The substitution technique enables searches for $f(A, B) = f(B, A)$ e.g. search for mathematical objects showing commutativity of multiplication. This is an important advantage. It also allows for searching independent of variable name, for example $\int x^2 dx$ will match $\int y^2 dy$. This has an impact on the subformula searching and can be currently solved only by adding all subformulae to the index. Currently it lacks the support for simple text searches limiting searches only to mathematical notations. The project has this issue selected as one of its future goals.

The mathematically correct behaviour has borrowed some techniques from theorem provers inheriting also their disadvantages. The searching is basically traversing of a formula tree from the root node to leaves. Without extensive evaluation it is difficult to estimate the seriousness of this issue but the first results show no serious performance drawback.

The search engine web front-end is very similar to the of `MathDex` but the query language is different. An extended form of `MathML` is used. Currently it can index documents in Content `MathML` and `OpenMath`format. The support for Presentation `MathML` is limited and thereby also limiting its usage. The reason is that the presentation does not include semantic data about the mathematical notation.

The question is if they can not meet their goals in real-world data (as by online publishers, WWW) where the diverse and heterogeneous mathematical structures presented can not be supported mathematically correctly because of lacking semantic information.

2.3.10 Summary

Presented solutions have one thing in common. All of them are applicable in specialised environments with specialised purposes with relative success. At this stage of development they can not be compared and evaluated. They greatly differ in technical quality and currently the practical application aiming for real data is very uncertain for most of them.

The database approach can be very useful in connection with a mathematical aware search engine. The CiteSeer search engine proves the success of a simple idea to split the scientific documents into several parts allowing them to be indexed separately. Theorem provers and proof checkers have the advantages of mathematical correctness and deduction at the cost of performance and the considerable dependence on the availability of semantic information. Not only the presence in the indexed documents but mostly also for the inputted query which must exactly specify (either explicitly or implicitly) in which mathematical structures it should be searched for, proved etc. Another disadvantage is that it is not flexible and does not allow for similarity search nor for text search. Nevertheless, in connection with semantic web this technique can be proved useful.

The mathematically most aware search engine is `MathWebSearch`. The advantage is clear but this brings also many disadvantages from the practical point of view. The main goal is very ambitious. It tries to correctly interpret the meaning of formulae. To behave mathematically correct it must heavily depend on the semantic information which results in decrease of precision and recall when operating over documents with little semantic information. The matching is done traversing special trees which can bring the problem of performance but it has not been confirmed yet. The user query should also include semantics (currently not supported) to fulfil the main goal. The lack of full text search makes this search engine very limited and the integration with a text search engine can be tricky. As the only search engine it supports mathematically equal notations with different variable names but at the same time it forms the question how to decide what is a variable and what is not in semantically poor documents.

The second most mathematically aware search engine is `MathDex`. It can search for both text and formulae. The mathematical precision is however unclear. The ranking algorithm uses techniques to increase the precision and relevance like frequency of subformulae occurrences. It also uses the paradigm of CiteSeer applied on formulae which are split into sections. The gain from this approach is unclear too. It depends less on the context than `MathWebSearch`. The user query does not need to contain much semantic information to exploit the power of the search engine which is from the mathematical equality smaller than of the `MathWebSearch`. But it definitely does not mean that the results must be of less or same value. The search for mathematically equal formulae is limited.

The last solution is `LeActiveMath` which uses special semantically rich format. The advantage of the special format is clear and can be very significant but the applicability for data on the WWW is dramatically decreased. The feature of storing depth information together with the formulae is very interesting but whether it is an advantage or a disadvantage is not clear. The user query language was aimed for common users. The real application of `LeActiveMath` in the `ActiveMath` system can provide the authors with very important statistics about common user of mathematical search engine.

It is difficult to create a search engine without knowing the most common use of it. The nonexistence of user expectations about mathematical search engines makes the first prototypes very diverse. This makes very difficult to decide which is the best. It is clear that the less semantic information is used the better it will handle documents on the WWW (till up to certain point because full text search engine are really not suitable for mathematical searching). Introducing axioms like commutativity to mathematical structures which do not support and also to structures where it holds can surprise the user. On the other hand it is clear that in most cases at least the recall will be improved.

The conclusion is that the main focus now should be aimed for the WWW - where the most mathematical documents can be found. In this stage the precision or recall is not as much important as the user feedback mainly because the recall and precision is very questionably in mathematical searching. Moreover many searches will be originated by a common user not possessing the knowledge about the item searched for and therefore to rely on the semantics of the user query is probably pointless and very limiting. Furthermore to rely on semantics on one side of the search engine and not on the other (e.g. only in the indexing phase) can produce unexpected results. Also many of the presented techniques, features are not yet proved to be of any use. And because of this, further "enhancing" can decrease the value of the search engine for a common user. The proposed solution tries to fill this gap between void and theoretical or highly specialised solutions.

2.4 Definition and characteristics

The analysis of available mathematical search engines from the previous section gave a very good impression on how this research field is enormous and diverse. The definition of a mathematical search engine accepted by the users has not been successful yet. One of the reason is the big number of different types of reasonable and usable questions we can ask a mathematical information system. Users want proofs, similar theorems, similar formulae, proof concepts, equal formulae, formulae based on output, etc. The following definition of important features puts stress on practical usage - usage over real word data as the WWW. It does not repeat the features every search engine should have like ranking algorithm. However, the general definition must be based on an exhaustive testing and feedback from the mathematical society.

The first fundamental characteristic is clear. It must support formulae searching and searching for other mathematical notations. The second fundamental feature is support of simple text search. Another important feature must be an extensive, clear but expressive query language with graphical user interface as an improvement but definitely not a replacement. It should be extensible as new forms of queries will be added.

A usable scientific search engine should be able to process the `pdf` format because it is the most common format for scientific publications. The few available converters can produce only output with the information about the layout (like Presentation `MathML`) not about the real semantics (like Content `MathML`). Therefore another feature should be support not only for semantic formats but also for presentation formats. Because there is a vast number of basic formula equal notations at least some kind of similarity search should be supported and the next feature immediately following is the support for subformula search. The atomic information unit must be chosen carefully.

Additional features can contain categorisation of documents, classifying theorems or formulae against a database. The result page should show parts of the document including searched terms and grouping of similar documents which can significantly assist in the user decision of relevance.

The summary of main features

- mathematical awareness

- similarity search and subformula search
- both content and presentation support
- easy, extensible but clear query language

2.5 Proposed solution

The design was based on the analysis of engines available at that time. None of the most important ones (`MathDex`, `MathSearchWeb`, `LeActiveMath`) have been available at that time. Although the new engines bring new solutions, there is no need for any change. The proposed solution solves the mathematical searching problem from the practical point of view. It is still a new contribution in this research field and proposes novel indexing technique.

The main design decision was whether to use an already existing type of a search engine and extend it or design a completely new one. The latter choice has lead always more to academic solution than to practical. Two of the presented engines copy or slightly modify already existing indexing techniques and it is highly improbable that in the near future another search engine type will be capable of indexing large amounts of data but the full text search engine. And moreover even full text search engines usually index only first few kilobytes of documents [20]. As mentioned in the introduction full text search engines can index text documents with little or completely without any metadata and still be very valuable by applying many sophisticated techniques, tricks and other solutions to gain the most information from a document and then use it in the ranking algorithm. The text search engine ranking algorithm works well on mathematical documents too because they are still documents. Each theorem is usually well described in the text. But it could work better. Mostly the marginal cases like completely new documents, which tend to rank lower, could be boosted after using the mathematical information making important new research results available.

The proposed solution is based on a full text search engine. Instead of reinventing the wheel an already existing solution is used. The implementation extends the Egothor [12] search engine.

A mathematical document can be separated into two different sections. The first one is the text section and the second one is the mathematical section. This way all the advantages of a full text search engine can be used with the possibility to limit the search by mathematical content. A search for a proof of a formula could result in searching for word "proof" in the text section and the formula in the mathematical section. When the search engine supports the proximity operator it can be even specified that the word "proof" and the text representation of formula must be at most N words apart. The proposed search engine considers mathematical notation as some sort of very important metadata of documents which are indexed and used in ranking.

To exploit all the existing features the proposed mathematical search engine is designed as an extension of a text search engine leaving all its features untouched. This way the theoretical background of the proposed solution can be implemented in any existing search engine. The only two changes required are: 1) the support of a mathematical query language in the front-end, 2) use the mathematical data when ranking documents. The second is not essential for the function of the search engine only for the quality of results. The design of the mathematical extension in the indexing phase is illustrated in Figure 2.1. The mathematical extension is a mediator between mathematical documents and the indexing part.

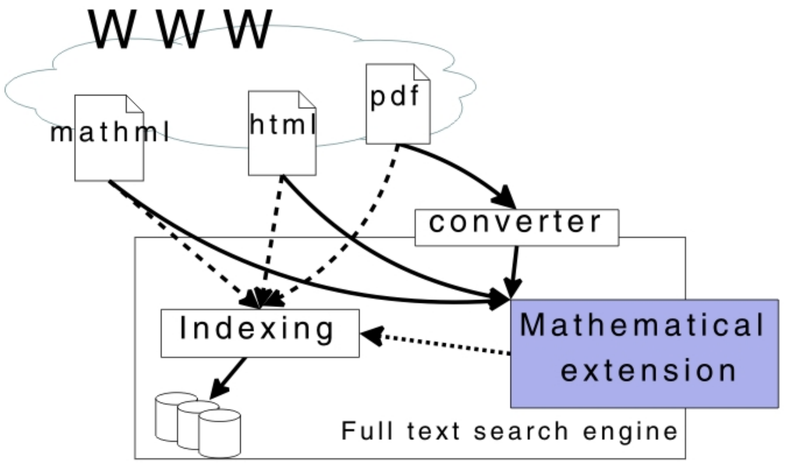

Figure 2.1: Design of the proposed mathematical search engine

The following sections address the problems identified above.

2.5.1 Extraction of mathematical content

Extracting and classifying content of a mathematical document includes both converting the document to a supported format and parsing the format to at least syntactically equivalent form.

Mathematical text is highly-structured and symbolic. Most common way of publishing a mathematical text is to write it in a typesetting language (TEX, LATEX) and then export it into either `pdf` or `ps` format [15, 16]. The first two formats can contain formatting information but many online publishers (e.g. ACM) remove it. It can be even impossible to extract the text from the raw file itself because the transformation function from character codes to glyphs[7] can be irreversible. This makes parsing very difficult, sometimes impossible and very time consuming [7, 15]. It is even more

[7]A glyph is a graphical shape [16]. For example, "a" or the dot on the letter "i" are glyphs.

difficult with mathematical expressions because some characters are not part of the standard encoding.

The problem can be solved by using:

1. man power - converting manually
2. OCR - optical character recognition. Most of the successful converters [8, 7] using OCR are tuned by many-year experience. But OCR only gives the character code and its position on a page and it must be further processed.
3. raw file processing - neither character decoding nor character order is guaranteed
4. combination of OCR and raw file processing - combining guaranteed character codes with formatting, when available, can be very accurate [42, 16]

Even when the source text can be translated to the most used LaTeX source code, converting it to a format supporting content markup like `OpenMath` or Content `MathML` is a very complicated problem [11].

Because digital documents are mostly published on the WWW another approach is to use an `HTML`-like language (e.g. `MathML`) to view mathematical documents. The goal of `MathML` is to enable mathematics to be served, received, and processed on the World Wide Web just as `HTML` has enabled this functionality for text. `MathML` value lies in the ability to encode both mathematical notation and content. It can be embedded in `HTML`. In the last few years the size of mathematical content using `MathML` has increased rapidly. However, it is still very seldom used as the first choice by online publishers.

After analysing several formats (`OpenMath`, `MathML`, LaTeX, TeX, `XML`, `pdf`, `ps`, `HTML`) which can contain either mathematical content or presentation of mathematics and due to this are suitable for mathematical indexing, `MathML` was chosen as the primary format. The reasons are that this format is sufficient for mathematical encoding, it can be embedded in `HTML`, there are many converters to this format and finally it is pushed by W3C Consortium.

Because the `pdf` format is by far the most used format special emphasis has been made to be able to index it. In June 2007 the Infty[8] Project [8] released product version of their OCR system which is specialised for documents containing mathematics. Using this program the `pdf` format will be converted to `MathML` which is already supported.

The implementation currently supports Content and Presentation `MathML` and LaTeX-like syntax.

Classifying simple text and scientific notation

The classification is one of the main tasks of the converter and is not part of the search engine. Simple heuristics can be used to decide whether a document includes

[8]http://www.inftyproject.org/en/software.html#InftyReader

mathematical content or not. Each document is processed by the search engine itself in the document format that the full text search engine supports. If a document contains mathematical notation it is converted to a supported format (`MathML`) and then delegated to the mathematical extension which extracts only mathematical formulae from the content. The content is marked as mathematical.

The techniques used to classify the document are beyond the scope of this paper.

2.5.2 Indexing and design of ranking algorithm

Text search engines parse documents to tokens which are simply speaking words and these words are then stored. It means that a mathematical formula must be represented by text consisting of one or more words. A query is just a comparison of words.

One of the goals is to support mathematically equal notations (e.g. search for $a * (b + c)$ should also return $a * b + a * c$). Another goal is the support of at least simple similarity and substitutional matching (e.g. search for $a+b$ should return also $b + a$). The following problems or limitations must be solved in the indexing phase.

Full text search engine limitations

The main disadvantage of text search engine is clear. In the original form it can search only for documents containing specified words and therefore only exactly equal string representations of mathematical notations would match. For a mathematical search engine to be usable the search for subformulae must be available. This implicates that one formula can not be represented by only one word but by one or more ordered sequences of words.

Mathematical

Summary of important issues:

1. no commonly used mathematical format nor unitary notation $1/x = \frac{1}{x} = (x)^{-1}$, $\pi = \Pi = Pi$, $xy = x * y, ...$, more character codes or names mapped to one symbol
2. symbol meaning dependent on context
3. structured text
4. no canonical form ($1 + 1 + a = 2 + a = b - b + a + 2$, $sin^2 x = 1 - cos^2 x$)
5. equivalent transformation rules - e.g. *commutativity*, *distributivity*, *associativity*
6. many mathematical structures with different axioms

Solution

To fully exploit the full text search engine and eliminate the main disadvantages the indexed formula is expressed not only in one but in more representations. This technique is called *augmentation*. It does not solve the unique canonical form problem, but it can greatly reduce the possibility that two equivalent formulae do not match. To store exactly all of the possible representations is clearly impossible because unique canonical form of mathematical formulae does not exist. Theoretically the number of textual representations of one formula is not limited. Due to the performance the maximum number of representation is set to a fixed number. The upper bound can be changed.

The second technique is called *generalisation*. The algorithm in several iterations computes a more generalised representation which is then indexed. The generalisation is based on a set of *transformation rules*. In each step one or more *transformation rules* are used. Precisely speaking some assumptions are made on the underlying mathematical model which is simplified in each step of the algorithm. This way the representation from later iterations match more formulae. It is clear that the first iteration is the most important and naturally it is ranked as the most important. Another problem is that mathematically equivalent formulae with the same but permuted operators and operands would be considered as different when comparing letter after letter. *Ordering algorithm* guarantees that these formulae would have the same representation. This can be guaranteed because of the simplifications and assumptions made on the underlying mathematical apparatus. Common indexing algorithm works only with single linear words whereas mathematical formulae can be structured in more levels (e.g. argument, exponent, factor). To adapt mathematical formula to normal text, *linearisation* must be performed. Each formula is converted into linear stream of text words.

Linearisation, *transformation rules* and *ordering algorithm* simplify the complex and highly symbolic mathematical structures into linear structures with strictly well defined symbols where searching is possible but at the cost of losing unambiguity of formulae.

To solve the first mathematical issue a very simple but important step must be performed before parsing the document. Symbols that do not have a representation in a standard encoding have to be converted to words. For example $\int$ is stored as int.

Mathematical symbols can be context dependent. As a matter of fact, most of them are ambiguous when standing alone. For example $\Pi(a+b)$ can be either a function or a multiplication of Pythagoras' constant. Because mathematical search engine does not store context results containing both the function and the multiplication are returned. It is left on the user to decide which result is relevant. The solution to these ambiguities is to eliminate them by specifying text query to include only documents either containing specified words or not containing specified words. It is very likely that there are not many documents where one symbol has more different meanings. This implies a very important statement. The meaning is not important when the context is not known. The solution is to choose one meaning and operate with formulae identically at both sides of the search engine (indexing phase and searching). Each ambiguous symbol is converted to its normal form, for example π, Π, Pi, pi to

π. The best specification of normal forms is specific to the data set and is a subject to evaluation.

Many scientific fields use formulae (physics, mathematics, computer science, medicine, chemistry, etc.) to describe various processes. Many formulae are sound and valid only in specific mathematical structures. There is a huge number of different mathematical structures. Instead of distinguishing between them, all structures are generalised into single one in which these basic and most common axioms hold

- **commutativity** - $a + b = b + a$
- **associativity** - $a + (b + c) = (a + b) + c$
- **distributivity** - $a * (b + c) = a * b + b * c$

The author estimates that these rules hold in most mathematical structures found in Internet. The rest consists of either structures which are not suitable for full text search or so specific that they can be easily found by common search engines.

The indexing phase is described in more detail in Chapter 3.

2.5.3 Design of the query language

The most common groups of users of a scientific search engine are students and researchers. It can be assumed that both groups have already came to contact with scientific documents. Scientific documents are written in LaTeX and therefore to reduce the time to understand the query language a LaTeX-like language is used. It can express mathematical notations fairly intuitive. The advantage of this approach is that the language is de facto standard. The disadvantage of this approach is that it lacks any semantic information but because the proposed index algorithm is not dependent on semantic information the disadvantage is eliminated.

The details of the query language and other important parts of the searching phase can be found in Chapter 4.

3 Indexing Mathematical Formulae

The most important part of a search engine is indexing and nearly all of the problems need to be solved there. The indexing phase can be split to two not necessarily isolated stages.

The first one includes the indexing algorithm and is responsible for parsing the input document to supported structures (words, sentences, paragraphs etc.) which are then converted to streams of text words. Words with additional information (information about the position in document, the value of relevance to the document etc.) are called document tokens and are the smallest information units of a document. The text word represented by a document token usually undergoes a process of modification and normalisation which produces one or more variations of it which are then stored in a database. The information unit is called `token`.

The indexing algorithm of a full text search engine must use a language recognition technique. It is responsible for analysing parts of the text and parse them correctly. The analogous technique must be also used in the mathematical search engine and is called *formula recognition*. After the formula is recognised it is transformed to several textual representations which consist of one or more words.

The second stage defines the ordering of results. Ordinarily there are many relevant results which are sorted by a ranking algorithm. The statistics show that search engine users do not look at more than twenty results and therefore the ranking algorithm is very important (AOL Research). In present days the most important search engines conceal their ranking algorithms not only because of competition but also because of spamdexing[1]. Currently, this problem is negligible in the design of a mathematical search engine.

There are many index data structures which can be used to store the data (inverted indices, suffix trees, citation indices, ngram indices, trees, ...). Some of them are also used in the scientific search engines described in the previous section. The most popular data structure is the inverted indices also because it allows full text search. All well known full text search engines use this type of data structure.

There are few very important factors which must be considered when designing the index phase: 1) merge factors, 2) storage techniques, 3) index size, 4) lookup speed, 5) maintaining the index over time, 6) fault tolerance. Even a brief description of all parts of a full text indexing phase is outside the scope of this master thesis and can be found in [1].

[1]Spamdexing or search engine spamming is the practice of deliberately modifying HTML pages to increase the chance of them being placed close to the beginning of search engine results.

3.1 First stage

The reasons for selecting `MathML` as the main supported format are described in Chapter 2. It supports both the Presentation and Content `MathML`. The example of `MathML` document format is shown in Table 3.1.

Content `MathML`	Presentation `MathML`
`<apply>`	`<msup>`
`  <power/>`	`  <mfenced/>`
`  <apply/>`	`  <mi>a</mi>`
`  <plus>`	`  <mo>+</mo>`
`    <ci>a</ci>`	`  <mi>b</mi>`
`    <ci>b</ci>`	`  </mfenced>`
`  </apply>`	`  <mn>2</mn>`
`  <cn>a</cn>`	`</msup>`
`</apply>`	

Table 3.1: Simple MathML markup example

The Content format copies the hierarchical structure of a mathematical formula and therefore can be naturally transformed to a tree. The presentation format is on the other hand very linear and it is ambiguous ($a\,b$ can mean also $a*b$). The approach to transform both notations to trees is followed in both cases. There are many tree types representing mathematical formulae but for the purposes used here the differences are not important. However, it must contain some additional information. The theory is built on a specific type of tree but basically any other structure can be used on which the *transformation rules* can be applied. One example of such a tree is in Figure 3.1. Because single leaf node of a tree contains little information the rules operate not with leaf nodes but with more complex structures called `formula tokens`. It is the same paradigm as used in full text search where the atomic information unit is a word and not a letter. It is the smallest mathematical unit which can be searched for. The difference is that words are common constructs of natural language whereas the `formula tokens` must be constructed artificially. The construction is parametrised by the formula appearance - number of operands, number of operators, number of depth levels etc. This part of the system responsible for building `formula tokens` from a formula is called `formula tokenizer`. It is crucial for the fine granularity of the mathematical search engine.

Each mathematical formula consists of operands and operators. Whereas operators are functions taking arguments, operands are divided into these categories: 1) numerical constant - $0, 100, -104, ...$, 2) common known constant - $\Pi, e, ...$ (Note: only few symbols are predefined as constants because also many common used sym-

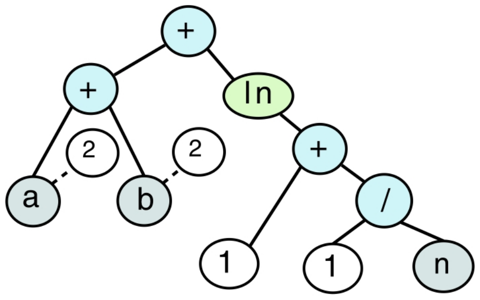

Figure 3.1: Formula tree for $a^2 + b^2 + \ln(1 + 1/n)$

bols mean something else in other mathematical domain), 3) common known operator - $\sum, \int, ...$ (Note: only few are selected as common and the reason is the same as in 2)), 4) unknown variable - $a, F, ...$

Definitions and notations used in this text are ordinarily used but some need to be redefined.

1. **Formula tree** - Undirected connected tree data structure in which each node has at most two children. Leaves which have children are called internal nodes. All other nodes are called leaf nodes. Internal nodes contain mathematical functions (+,/,ln, ...). Leaf nodes can contain numbers, constants and variables. Each node can have one or more mathematical subformula connected to it like index or exponent. The depth of all nodes must be stored. It is computed with the respect to the basic mathematical operators. For example $x + y + 10/x$ will have three levels, with +,+ as root nodes. x, y, / as child nodes and 10, x as children of /. The depth must satisfy a condition that at the same depth level the commutativity axiom holds.
2. **Operator** - Internal node of formula tree.
3. **Operand of an operator** - A child of an operator.
4. **Most important operators** - All nodes with the least depth level.
5. **Most important operands** - All operands of the most important operators.
6. **Canonical representation of formula** - Result of the *ordering algorithm* used on a formula tree.

From the mathematical point of view, in this stage a function Q is constructed. The domain is the space of all mathematical formulae (F) and the range is $F^N := F_1 \times F_2 \times, ..., \times F_N$. Simply speaking it produces N (N is a predefined constant dependent on the *generalisation rules* and *transformation rules*) formulae $f_1, f_2, ..., f_N$ for one input formula. The function Q can be defined: $Q : F \to F^N$, $Q(f) =< f_1, ..., f_N >$. The function Q must satisfy one requirement about its domain F^N: for all valid i, F_{i+1} is a generalisation of F_i. It means that F_{i+1} is more simple (does not contain so many axioms, more symbols are mapped to one) but it may still contain new symbols. The algorithm which produces the output of function Q is called *generalisation algorithm.* Different representations are not mathematically equal but the advantage is that more and more formulae are mapped to the generalised formula in each step. Practically speaking the output of the first stage of the indexing phase of the mathematical extension is one or more different textual representations of the input formula. The representations consist of products produced by the *generalisation algorithm* with each representation having its advantages and disadvantages. Advantages and disadvantages of chosen representations are discussed in Section 3.1.4. Finally the *ordering algorithm* orders the operators to ensure that mathematically equal but permuted formulae are transformed to the same representation.

One important feature is that there can be more than one function Q with different specialisations. The only limitation is the number of important formulae in one document. This number is usually negligible comparing to the number of text words.

Structure of the *generalisation algorithm* is not strictly set and can be changed to meet special demands. In each step one or more *transformation rules* are applied and a more general formula is constructed. The construction of f_i can be described by defining functions Q_i which take a formula and produce the $i - th$ representation by applying the rules at this level. The main algorithm used in the mathematical extension performs these steps:

```
Input: Input formula - $f$
Output: $f_1, f_2, ..., f_N, f_i$ - formula
$f_0 := f$;
for $i = 1$ to $N$ do
    $f_i := Q_i\ (f_{i-1})$;
    $f_i :=$ ordering $(f_i)$;
end
```

Algorithm 1: Main algorithm

The resulting set must be converted to text. Text is an ordered set of words. Infix form is the common notation of mathematical formulae. The advantage is the readability but the disadvantage is the necessity of parentheses. The solution is not to use infix. The proposed search engine uses reverse polish notation also called postfix notation. This technique has two main advantages. The first one is that it does not need parentheses and the second one is that it enables one type of similarity searching as illustrates the following example. The formula $(a + b) - (c + d)$ is converted to $ab + cd + -$, let's assume that formula tokens are $ab+$ and $cd+$ and so the index database contains three words in this order: $ab+$, $cd+$, $-$. This allows to search for the subformulae ($ab+$, $cd+$) without knowing the operator between them.

The definition of a *formula token* is very difficult because it depends on many aspects like user behaviour, mathematical context, data set. The vague definition is: the smallest part of a formula which can be searched for. The precise definition is a subject to evaluation. A model definition can be found in Chapter 5.

3.1.1 Transformation rules

List of *transformation rules*:

1. **Partial evaluation at each level**

   Example: $7 + a + 5 \rightarrow 12 + a$

   See rule *#2*.

2. **Approximate numerical constants**

   Example: $3.14 \doteq 3$

   This approximation does not practically decrease search precision. Formula either contains important special constants (because they are special they should be easily found on the WWW by any full text search engine) or does not contain an important constant and can be approximated. Other cases are negligible.

   In many parts of mathematics formulae with different constants are still considered to be very similar and can be easily transformed or immediately used when found. This is the reason why most of the constants can be replaced by symbols without effecting the search. However, few exceptions should be taken into account (see rule *#6*).

   To keep the possibility of a nearly perfect match (with constant approximation imprecision) constants should not be immediately replaced. At least the first produced representation should contain them.

   The approximation is parametrised by rounding behaviour (mode - up, down, floor, ..., precision - integer, float, ..., etc).

3. **Remove brackets using distributivity**

   Example: $a * (b + c) \rightarrow a * b + b * c$

   The general intention is to linearise formulae. It is clear that one formula has more different textual representations (but mathematically equal) with the possibility to use brackets then without. This implies the statement that the more two mathematically equal formulae are linear the bigger chance is to convert them into the same canonical representation. In general the *ordering algorithm* is more suitable for linearised formulae.

   This transformation is parametrised by the appearance of their operands. It is the same problem as with *formula token*. Complex operands are less suitable for the distribution than simple operands. The appearance of the result can be considered very different to the original formula and therefore the precision would be very low. By adjusting the requirements for the operands the precision can be kept above a specified limit.

4. **Multiply tokens**

   Example: $\frac{a+b}{2} * \Pi \rightarrow \frac{\Pi a + \Pi b}{2}$

Apply this rule only when one of the tokens is a division with numerator being an addition or subtraction. The reasoning for this transformation is the same as with *#3*. When the number of transformation applicable to a formula is cut down less different textual representation of a formula can be created and the chance to match them is higher. In this case the prohibited transformation is to take out and operand.

This transformation is also parametrised by the appearance of their operands. Complex operands are less suitable for the multiplication than simple operands.

5. **Assign each numerator its own denominator**

   Example: $\frac{\Pi a+\Pi b}{2} \rightarrow \frac{\Pi a}{2} + \frac{\Pi b}{2}$

   Each numerator *token* will be detached from the original division if either $+$ or $-$ is between the numerators. The reason for this transformation is the same as for *#3* and *#4* to limit the number of possible representations of a formula.

6. **Replace constants with *const* symbol**

$$\begin{array}{rl}\text{Example: } 74 + a^2 + b^2 & \rightarrow const + a^{const} + b^{const} \\ \text{or} & \rightarrow const + a^2 + b^2\end{array}$$

   As mentioned in *#1* it is meaningless to distinguish between formulae only by value of a constant.

   An extension of this rule can contain the requirement that standalone constant on one depth level are not replaced as in a^2: $7 + a^2 \rightarrow const + a^2$ because they can be considered very important. Another extension can disallow replacing known constants as Π, e, etc.

7. **Replace unknown constants and variables with *id* symbol**,

$$\begin{array}{rl}\text{Example: } a^2 - b^2 - c^2 + 2bc\cos\alpha & \rightarrow id_1^2 - id_1^2 - id_1^2 + 2id_1 id_2 \cos id_3 \\ \text{or} & \rightarrow id_1^2 - id_2^2 - id_3^2 + 2id_1 id_2 \cos id_3 \\ \text{or} & \rightarrow id_1^2 - id_2^2 - id_3^2 + 2id_2 id_3 \cos id_4\end{array}$$

   Variable names can differ. Search for $x + y$ should also return $a + b$ or $R + S$. As mentioned in the beginning of this chapter mathematical structures are very context dependent and only few symbols are really standardised. Due to this fact unknown constants are considered as variables. Both are replaced by *id* symbol. This rule ensures to return also formulae with other variable names at the cost of precision.

   This approach has one disadvantage when searching for subformula. The variables can not be numbered globally because the position of subformula can vary and so the numbering in the input subformula is not known. Assume that a global numbering has been performed and these are representations of indexed formulae in infix order for readability and that each operand is a *formula token* for simplicity: $id_1 + (id_2 - id_3)$ and $1 + (id_1 - id_2)$. Logically, both match the search term $a - b$ when searching for subformula. In the first case it should be converted to $(id_2 - id_3)$ and in the second case to $(id_1 - id_2)$. Because of this the global numbering can not be performed and so the precision lost at this level can be significant. Therefore, this rule is applied at the end.

An extension to this rule follows from the definition of a *formula token*. *Formula token* is the smallest searchable unit and therefore a numbering inside a *formula token* does make sense and should be performed as shown in the second and third possible example transformation.

The list of the rules applicable to a formula is not complete but contains rules solving the most common mathematical problems. Any other rule can be simply added. The number of text representations of one formula is not limited and depends on the goal of the engine.

The searching algorithm tries to find a match at the most exact level and if it does not find any result it can try the more general level. In the manner described even the most generalising rule can have impact on the recall and precision only at its own or lower level. Therefore these rules should be put at the end.

More technical simplification rules

Because the formulae are very complex non linear structures more simplifications must be introduced. All the following simplifications are technical. This means that there is no reason for their presence other than keeping the mathematical extension as simple, but still usable, as it can be. Equations are considered as two formulae with equal operator between them. Constraints and variables of known operators are not considered ($\int_0^\infty x^2 dx \rightarrow \int x^2$). These simplifications are not consequences of any limitations and are introduced only to allow the mathematical search engine to be very simple. They are subjects for further improvements.

3.1.2 Ordering algorithm

The purpose of the *ordering algorithm* is to guarantee that two mathematically equal formulae with the same but permuted operands have a unique canonical representation and that two similar (but not equal) formulae have a similar unique representation.

One definition of formulae similarity is that the more operations they have in common the more similar they are. The same can be imagined as comparing the formula tree using breadth search. To get the best search results the token ordering must be done with regard to the similarity of operations not to the difference of characters or values. The ordering does not use simple string comparison, if not really necessary, but a rather more complex comparison algorithm.

The usual operator priority must be kept, otherwise canonical formula and original formula do not need to be equivalent. The basic operator priority from least important to most important is $+, -, *, /$.

The *ordering algorithm* is used always on one depth level at a time. Assume a formula tree has more than one most important operator. Then the *ordering algorithm* uses this order:

1. smallest numerical constant (more constants can be at one level, despite the first *transformation rule*, when the operator between them is unknown or not suitable for arithmetic)
2. common constants with predefined order (e, Π, ...), if equal use operator priority
3. known operators with predefined order ($\int$, ln, ...), if equal use operator priority
4. other subformulae are ordered according to this rules:
   (a) ordered by operator priority - first $+x^2$, second $-x^2$
   (b) ordered by operand count - the operator with the least operands is at the beginning
   (c) if the operator priority and the count of operands are equal for two subformulae then order them recursively according to these rules:
      i. priority of the first different operator
      ii. if exactly one contains numerical constant then it is the first one
      iii. if one contains more common constants then it is the first one
      iv. if one contains more known operators then it is the first one
      v. otherwise use string comparison

3.1.3 Proof of the ordering algorithm

The *ordering algorithm* must satisfy two conditions. Firstly, two mathematical equivalent formulae with the same operands but permuted using commutativity must have the same unique canonical representation. And secondly, two not equivalent formulae or equivalent but with different operands must have a different canonical representation.

Because *ordering algorithm* neither deletes nor inserts operands or operators the second condition is clearly met. If one formula contains operand or operator which the other does not the two can not have equivalent representation. Commutativity is mathematically equivalent transformation and therefore two not equivalent formulae will have different representations.

Schematic proof of the first condition is done using mathematical induction. The proof is built on formula tree depth. Firstly, prove that after applying the algorithm to leaf nodes of equivalent formulae results in the same representation and then for the $k+1$ level. Because input formula has finite number of operands and each operand is ordered by the algorithm exactly once the algorithm is clearly finite too.

- For $depth = 1$, meaning just symbols from the leaf nodes are being ordered and it is clear that the order is unique because no two different symbols are in the same category. If the same symbol can have more meanings only one is used as described in Section 2.5.2.
- Assume the theorem holds for $depth = k$.
- For $depth = k + 1$, because each child of an operand is already ordered both subformulae have the same operands. Because the same algorithm is applied on both formulae the ordering is the same.

When the depth is equal to the formula tree depth all single operands are ordered and so is the formula.

3.1.4 Formal description

A formal description of the algorithm:

```
Input: Input formula - $f$, set of transformation rules $T_1, ..., T_7$
Output: Set of formulae $f_1, f_2, f_3, f_4, f_5$
$f_0 := f$;
$Q_1 := T_1, T_2$;
$Q_2 := T_3, T_4, T_5$;
$Q_3 := T_6$;
$Q_4 := T_7$;
$f_1 :=$ ordering $(f)$;
for $i = 1$ to $4$ do
    $f_{i+1} :=$ ordering $(Q_i\ (f_i))$;
end
```

Algorithm 2: Generalisation algorithm

The textual description of the algorithm:

1. apply the *ordering algorithm*
2. store actual representation
3. apply *1.* and *2. transformation rule* - partially evaluate numerical constants on every level and round them
4. apply the *ordering algorithm*
5. store actual representation
6. apply *3.*, *4.* and *5. transformation rule* - linearise the formula which means try to minimise formula tree depth
7. apply the *ordering algorithm*
8. store actual representation

9. apply *6. transformation rule* - replace constants with *const*
10. apply the *ordering algorithm*
11. store actual representation
12. apply *7. transformation rule* - replace variables with *id* symbol
13. apply the *ordering algorithm*
14. store actual representation

The purpose of the first representation is to allow mathematically most precise match. The second representation is again very likely mathematically equal to the original form. It can be still considered very similar and therefore the first two representations have relatively high rank. The third representation resulted from more complex transformations which are parametrised by several factors. The main goal was to cut down the number of possible representations mapping very common variations to single one. The fourth representation uses the idea that constants are not important in many parts of mathematics. The last one limits the similarity to the operations and number of operands. It is probably the most severe transformation because there is no global numbering. However, local numbering is possible and can be very effective. On one hand the last representation allows searches for $a+b = b+a$ but on the other hand at the cost of a big precision lost.

3.2 Second stage

The ranking algorithm is taken over from the underlying full text search engine. Each word in a document has a weight which indicates the relevance to the document. For example the words in the title are ranked higher than the words in the body of a document. There are many heuristics which slightly enhance the relevance of words in the relation to the document.

Because the mathematical search engine produces words they must be ranked too. If two documents match($\doteq$) a search term but both match with different representation, $f_i \doteq f_j$ but $i \neq j, i < j$, then the document with f_i should rank higher. Let R be the rank of a word then the requirement for the ranking of the formula words can be written as: $R(f_1) \geq R(f_2) \geq ... \geq R(f_N)$. The mathematical background for this approach is taken from similarity searching and is described in Chapter 6.

3.3 Case studies

Case studies of formulae transformations are in Appendix A.

4 Searching Mathematical Formulae

The last phase of a search engine is the only user interactive phase. The user must formulate a search query which specifies her searching needs. Because the query language is usually very simple queries can be ambiguous. Statistics show that most text search engines users do not use the offered potential very much but are limiting the search string only to simple words.

The user interface of a full text search is standardised as can be seen in the most successful search engines (http://live.com/, http://google.com/, http://yahoo.com/). Also the presentation of results is very similar. The first few (around twenty) results are the most important and the results are sorted according to a ranking algorithm.

The searching phase must perform several steps: 1) query parsing, 2) mapping query operators to supported internal constructs, 3) find all words/phrases in the query, 4) evaluate the logic of the query and collect suitable documents, 5) sort them according to their rank and relevance to the query, 6) return the result list. Query constructs usually include boolean operators *AND*, *OR*, *NOT*. The mathematical extension must be part of 1) and 2) and the changes must be appended to the text search using the available internal constructs.

From the practical point of view a dramatic change to the interface could have great impact on the usability. Because of this no new graphical interface is designed but rather the common interface is extended to allow mathematical searching. The entry point to a full text search engine is a text field where a simple string is specified. The string is sent to the server which in return sends the result. The entry point to a mathematical search engine must be augmented by one or more fields where formulae can be expressed. The best solution from the short time perspective is a graphical application assisting in the creation of a formula but from the long time perspective a simple string query language is necessary. In both cases the query must be somehow encoded and sent to the server. In the latter case the inputted string can be send directly. In the first case the graphical representation can be converted to text too but the problem is that the query string can be very complex and because it is converted automatically it can be less user friendly. The most important goal is to have the query language as simple and user-friendly as possible but with no limits to the expressivity.

4.1 Query language

Most users searching for mathematics have already made contact with science papers. One assumption can be made about the diverse group of users. One of the programming languages that have been seen by most of the users is LaTeX. To allow intuitive usage the search query language proposed in this study is very similar to LaTeX. The main advantage is the standardised use. The main disadvantage is that it does not allow semantic information. However, the proposed search engine does not use the semantic information and therefore this disadvantage is eliminated.

Basic mathematical notations and operations must be supported. One basic ambiguous simple grammar that should be at least supported is illustrated in Listing 4.1. Terminal can be rewritten to supported constants, symbols, numbers or unknown symbols. Supported symbols must be carefully selected to avoid high equivocation and must match symbols in the indexing phase. The grammar is ambiguous but as mentioned above the lack of semantic information from the user and the lack of semantic information in the document enables us to select one specific meaning of the formula without having a negative effect on the search engine performance. The only requirement is that it is the same in the indexing phase as in the searching phase. Few query language search examples are in Table 4.1.

Despite the fact that the grammar is very simple it can be used to express most of the tested mathematical notation. More complex definition are either very ambiguous or too complex to try other technique but perfect match. The latter case depends on the amount of the specification which can be adjusted according to current dataset.

On this special type of ambiguous grammar similar techniques to *island parsing* can be used [17]. The grammar can be imagined as a sea which is divided into islands surrounded by water. Water contains the less important production rules and islands are the more important production rules. The rules are selected according to a predefined order.

```
-> S
S -> expr
S -> eq
expr -> expr '+' term | expr '-' term
expr -> term
eq -> eq '=' expr | expr
term -> term '*' atom | term '/' atom
term -> atom
atom -> atom '^{' expr '}' | atom '_{' expr '}'
atom -> (expr) | terminal
atom -> '\sin' atom | '\cos' atom | ...
terminal -> NUMBER
terminal -> DEFINED_CONSTANT | DEFINED_SYMBOL
terminal -> UNDEFINED_SYMBOL
```

Listing 4.1: Sketch of a query language grammar

Query	Meaning
a^{2}+b^{2}	$a^2 + b^2$
\Pi(a+b)	$\Pi(a + b)$
\int e^{-x^{2}}	$\int e^{-x^2}$
\lambda +y = \Pi * r^{2}	$\lambda + y = \Pi * r^2$

Table 4.1: Query examples

4.2 Performing query

The mathematical search engine computes representations using the *generalisation algorithm.* It is clear that it must be the same *generalisation algorithm* as in the indexing phase. If there is more than one representation it is appended to the text search query string using *AND* operator one at a time. This ensures that only documents containing both search terms are returned. The text query must be present in the text section of the document and the formula in the mathematical section. So instead of one query there is up to N queries. One query is built at each step of the *generalisation algorithm.* One advantage of this approach is that if there is a match at a step of the algorithm it is very likely more relevant than that the results obtained by performing query using representations from later steps. Therefore the queries are performed till the first match of K different documents are found.

The ranking algorithm of mathematical formulae is based on the similarity search which uses formula distance to rank each formula. The mathematical background is discussed in Chapter 6. The user can use this technique to limit his search only to a specified domain of similar formulae and achieving finer mathematical granularity.

The process of searching is demonstrated in Figure 4.1.

4.3 Displaying results

There are many ways how to display full text results. Usually, very helpful is to display parts of the text where the word/phrase was found. It is called highlighting and helps the user to decide which document is relevant without the need to open it.

The problem with mathematical documents is their presentation format as described in Chapter 1. There is no effective technique which extracts interesting parts from found documents without big storage overhead or undergoing the same process as in the indexing phase which is very time consuming. Another problem is that the formula representations stored in the index are different than the original formula and there is no connection between the original formula and the representation except the position in the document.

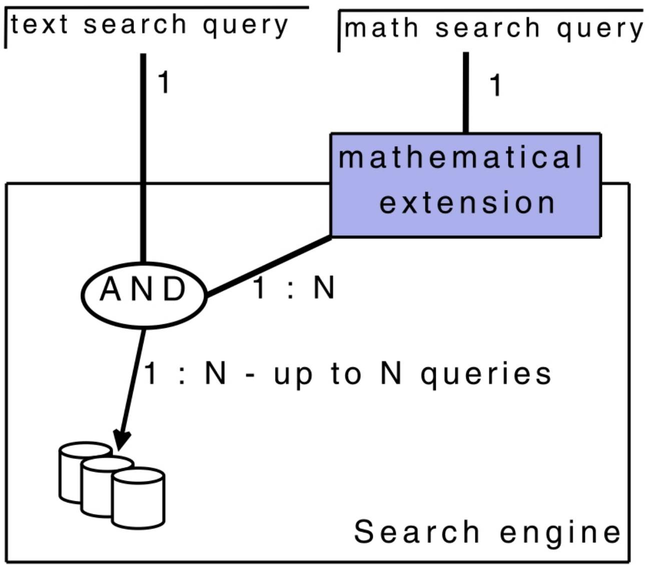

Figure 4.1: Query flow chart

The searching phase of a mathematical search engine must display at least a small abstract of the text extracted from the document. Because of the common structure of scientific texts proper authors and titles should be displayed too, if available. All other features are optional.

5 Prototype Description

The proposed mathematical search engine is designed as an extension to an existing search engine. The prototype implementation is based on the Egothor v2[1] search engine.

The goal of this chapter is not to offer an extensive technical description of the prototype implementation but rather to describe parts of the system which had to be reimplemented and added to enable mathematical searching. The user manual and the technical documentation can be found on the enclosed CD in Appendix B.

The Egothor project was created because no other open project met specific requirements put on the full text search engine. The most important demands were put on the transparent handling in local and distributed environment and extensibility which allows to implement arbitrary searching technique.

It is implemented in Java language because of its native Unicode support, easy code management and high-quality support for networking. The Java language has already proven itself in Lucene[2] search engine.

The first goal was to design and implement a robust, efficient crawler and its supporting utilities. Then the WWW content can be analysed and the information gained used to improve the overall design. However, this strategy has had an impact on the current status. Several parts of the search engine has not been finished yet.

Egothor is highly configurable and the mathematical extension follows this goal. In the following sections only those parts are described which are important for understanding the implementation of the mathematical search extension. Other parts are described in [12, 35].

5.1 Indexing

The purpose of the indexing part is to create an index over a set of documents. The index is called *tanker*. After the index is created the most important operation will be adding new information. The index directory contains metadata with their index, index map mapping local unique identifiers of documents to global identifiers, then inverted index itself, actual occurrences in documents and term dictionary with quick access map.

[1]The project homepage is http://www.egothor.org/.

[2]The project homepage is http://lucene.apache.org/java/docs/.

Index is filled with information units represented called *index tokens*. It contains the text it represents and additional information about the token type, start, end of line and column of its place in a document, weight, number of sentence and finally relative offset. The token type can be, for example, `"<WORD>"`, `"<GRAM>"`, `"<NUM>"`, `"<DIGIT>"`, `"<MARK>"`, `"<DATE>"`, `"<LETTER>"`, `"<ACRONYM>"`, `"<BITMAP>"`.

The last mentioned token type can be used to add a characteristic or a feature to a document. A characteristic can be the language used, file type, year of change etc. In the implementation it is used to mark all documents containing mathematical notations. Furthermore, the token types are also used to distinguish mathematical notation from normal text. This way the desired behaviour is achieved to limit formula searching only to the mathematical section.

After the document is parsed to *index tokens* it is added into a filtering chain. The task of the filtering chain is clear - create new stream of *index tokens* by sending each *index token* through a set of filters. Egothor has several filter classes with clear purpose - `LowerCase`, `StopFilter` etc. This paradigm is very important and will be described in more detail below.

5.2 Searching

The searching runs over the inverted index and is performed in these steps: 1) analysis of the input query and splitting the query to words, 2) inverted lists of all words are found in the index (more precisely in the dictionary), 3) the logic of the query is evaluated over the words with hit relevance, 4) first K (arbitrary constant) hits are returned. The query evaluation model can be one of boolean, vector or fuzzy model.

The analysis of the query transforms the string representation to internal tree representation. The default operation in the root node is AND. All supported operators are mapped to one of the classes in `query.representation`: `QTerm` - word occurrence, `QProx` - proximity operator ([]), `QPhrase` - whole phrase occurrence (`""`), `QOr` - (`||`),`QNot` - (-), `QAnd` - (&&), `QGroup` - default behaviour is that it requires the occurrence of at least one its children.

5.3 Changes

Several modules were reorganised to allow straightforward adding of the mathematical extension. The implementation of the extension itself required only modifications in the indexing and searching phase. The next sections discuss these changes in more detail.

5.3.1 Indexing

Egothor v2 supports only `HTML` document format. The first change was to add support for Presentation and Content `MathML` document format. Because the format is based on `XML` standard Xerces `XML` parser[3] was used.

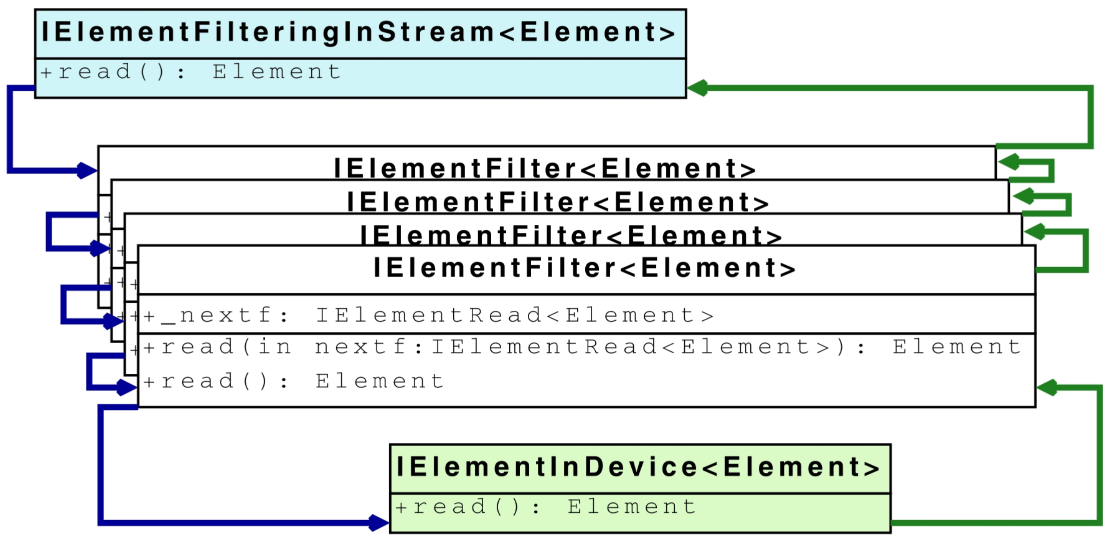

Figure 5.1: Filter architecture

The parsing itself is straightforward though it is very complex. The evaluation phase showed that the parsing must be very robust and flexible otherwise the whole formula can be invalid and besides can degrade the document value. Another important factor is the mapping of `MathML` symbols to known symbols. When a formula contains unknown symbol it can not be searched for. Moreover, the Presentation `MathML` does not contain information about the semantic meaning. For example `<mi>J</mi><mo/><mi>o</mi><mi>e</mi>` can be interpreted as $J * o * e$ or simply three tokens $J\,o\,e$. The problem illustrated in this example is very frequent and must be handled together with many other similar problems. Because of its complexity the parsing of `MathML` is located in a subproject together with the `MathML` interface provided by W3C Consortium.

The most important goal in the implementation was to add the mathematical extension to the Egothor transparently to demonstrate the advantages of the solution. The best choice was to use the filtering architecture. Before doing so it had been reimplemented.

The final architecture slightly resembles Boost filtering streams[4]. The basic idea is that a stream contains one or more filters followed by a device. Each stream can be either input or output. The Egothor project uses only input stream mode.

The common practise by the input mode is to have the device produce real data, for example, from a file. When a read operation is called on the stream the call is

[3]The project homepage is http://xerces.apache.org/.

[4]Boost is a C++ library already included in the C++ Standards Committee's Library Technical Report (TR1) and will very likely be part of a future C++ Standard. The boost iostreams can be found at
http://www.boost.org/libs/iostreams/doc/guide/filtering_streams.html.

delegated to the first added filter, which in turn calls the second filter etc. till the last filter calls the device which provides the filter with real data. Then the filter transforms the data and returns them to the previous filter etc. till the first filter transforms them and returns them to the stream. The schema is illustrated in Figure 5.1. The implementation of the idea of filtering stream is described in Appendix C.

This approach is easy to use and it enables adding *index tokens* produced from mathematical extension transparently by creating new filter class. After submitting all *index tokens* created by the original search engine the mathematical ones are submitted. The indexing algorithm does not need to be changed.

The filter class is initialised by a list of formulae which are converted to *index tokens*. The *formula recognition* technique used to build formulae uses either static symbol specification or dynamic specification illustrated in Appendix C. The evaluation phase showed that this is a crucial part of the system. It ensures that two identical formulae but stored in different encodings are recognised to be identical (Φ, ϕ, \u03C0, \u03A0, Phi, phi must be converted to one symbol). The mapping file has been created over a document set described in Chapter 6. First the formula is parsed to tokens, then converted to postfix notation and finally the tree is built. Along this process several helper techniques must be applied to eliminate possible problems (e.g. Joe problem). Again the solution is to select one approach and use it in both phases - indexing and searching.

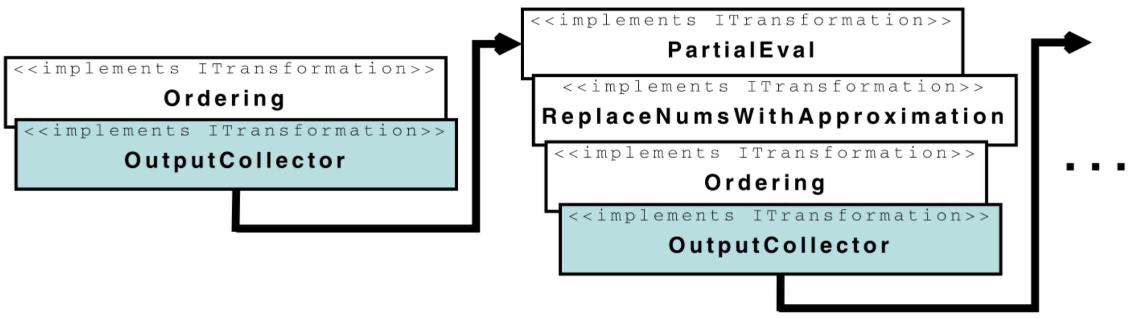
<<implements ITransformation>>
Ordering
<<implements ITransformation>>
OutputCollector
<<implements ITransformation>>
PartialEval
<<implements ITransformation>>
ReplaceNumsWithApproximation
<<implements ITransformation>>
Ordering
<<implements ITransformation>>
OutputCollector
...

Figure 5.2: Filtering chain

The overall algorithm is a list of transformations applied on a formula with one special transformation called `Output Collector`. This class collects the formula representations from each step of the *generalisation algorithm*. These representations are then split to words forming, with additional information, the *index tokens*. The splitting of representations to words is very important. The design relies on generic specifications which decide whether a subformula satisfies a specification or not. This part of the implementation is responsible for defining the atomic search unit for the mathematical search engine.

Each transformation has its own implementing class which operates over the formula tree. In this method the actual transformation is performed. The algorithm flow chart is shown in Figure 5.2. Note that the number of algorithms applied to a formula can be greater than one and is specified either dynamically or statically. The dynamic specification is in illustrated in Appendix C.

5.3.2 Searching

The same *generalisation algorithm* is used in the searching part. Each representation is appended to the text query with the AND operator forming the final query. If a query does not return any results the next representation is used. The internal representation is illustrated in Figure 5.3.

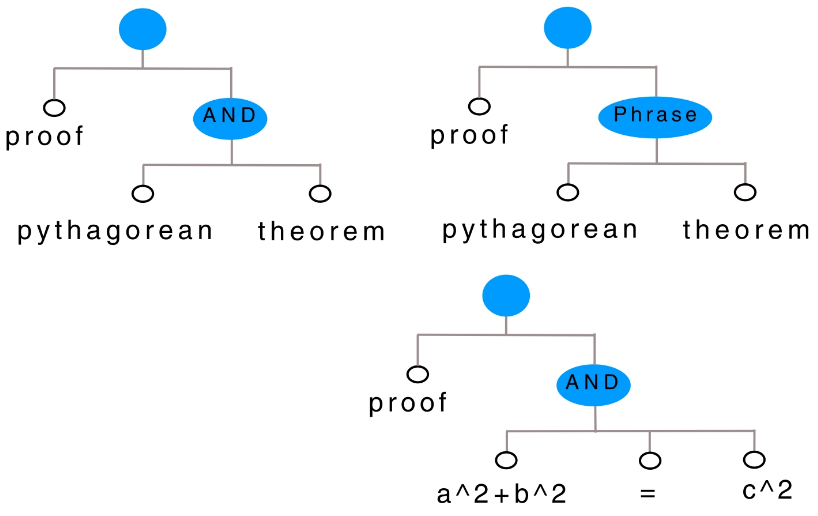

Figure 5.3: Internal query representation

The prototype implementation of the searching phase supports only command line user interface. It is clearly sufficient for evaluation purposes and moreover can be used for batch processing.

5.4 PDF support

The Egothor v2 project with mathematical extension supports embedded `MathML` into `HTML` and self-contained `MathML` documents natively. It can also support all mathematical document formats which can be converted to `MathML`. The goal of the implementation is to evaluate mathematical search engine and therefore it is focused on scientific notation rather than on normal text. The `pdf` support aimed for simple text is implemented in Egothor v1 using `xpdf`[5] utilities and can be integrated to version 2 easily.

[5]http://www.foolabs.com/xpdf/

The architecture of `pdf` support is illustrated in Figure 5.4. In the first step the `pdf` file is converted to monochrome portable pixmap graphical format (`pbm`). Then it is converted to tagged image file format (`tiff`) and finally it is converted to embedded `MathML` using OCR system. The usability of this method is discussed in Chapter 6.

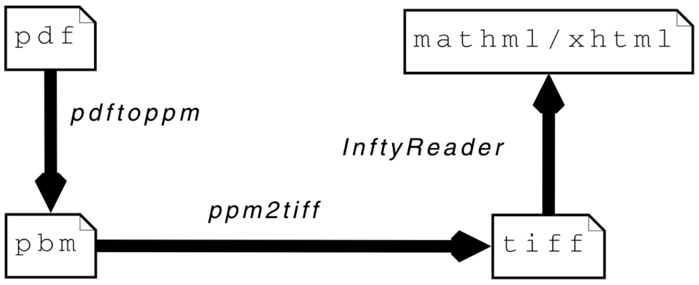

Figure 5.4: PDF conversion

6 Evaluation

Mathematical searching is a new research field with few practical results that is why there is not a widely accepted evaluation metric or method. Very few papers address this issue [18]. There are no testing databases for mathematical search engines like Trec[1] or Wang[2] with already predefined inputs and expected outputs.

There are three ways how to evaluate a mathematical search engine. One approach is to use existing techniques from general searching or information retrieval, another to use specialised techniques developed for specific purposes and the last one is to develop new techniques. The evaluation here exploits all three approaches.

The goal of this Chapter is to describe the important characteristics of this scientific search engine rather than to develop a new universal evaluation technique. It is very likely that the completely different approaches taken in the field of scientific searching can not be even reasonably unified.

The used datasets are described in the first section. It is very important to know the precise characteristics of the underlying datasets. The evaluation itself can be divided into several characteristics. Each characteristic describes one or more aspects of the search engine.

Fine granularity of the mathematical search engine is described in Section 6.2. This characteristic is important as it shows the effects of changed granularity on performance. It has an impact on the storage, speed and mainly on the usability where it defines the size of an atomic information element.

The most common metrics used to evaluate the performance of an information retrieval system, recall and precision, can be used too. But they would be rather less informative and more subjective than by, for example, full text search engines. However, the next characteristic in Section 6.3 is based on them and shows the search engine performance.

The size and speed of the implementation are shortly described in the next section. As there is nothing to compare it with and as it is a prototype implementation it is more informative than precise.

The last characteristic is similarity of the input formula and the indexed formula. Because this search engine is based on a full text search engine nothing is computed dynamically. The similarities are precomputed and stored as meta information. It means that each formula in the index database has a similarity value which is not dependent on the user query formula. The searching phase just selects the formula

[1]http://trec.nist.gov/

[2]http://wang.ist.psu.edu/docs/related.shtml

from the index and uses its similarity value.

The evaluation is closed by a short description of `pdf` converting process evaluation.

6.1 Dataset

Connections[3] (referred to as CNX in the remainder) is a site where educational material is shared. It is organised into modules, in other words small pieces of knowledge, which are further organised into courses, books etc. One of the basic categories is Mathematics and Statistics. The modules are displayed using embedded `MathML`, both Presentation and Content `MathML` are included in the document. There are around 500 modules in CNX Mathematics and Statistics section. CNX datasets have been obtained by partially mirroring its site in July 2007. The selection of this source is not arbitrary but it is the same dataset as indexed by *MathWebSearch*.

The first dataset, referred to as `CNX1`, is the whole set of files downloaded by the partial mirroring. This set also includes documents not containing mathematical formulae. The second dataset, referred to as `CNX2`, includes 16 modules from `CNX1` representing various mathematical research fields. The most interesting fields included are: Bilinear Transform, Torque about a point, Discontinuities, Interesting Graphs, Finding the Domain of Radical Functions, Basic Mathematical Operations, Angle between vectors, Basic Definitions in Stochastic Processes, The Central Limit Theorem, Important Examples of Expectation, Fixed Point Arithmetic, Fourier Analysis in Complex Spaces, Functions of a Matrix.

Several abbreviations have to be defined before presenting the statistics. The definitions are in Table 6.1. For example #0.Es means element count in the original formula.

#	- count	**A.**	- average
D.	- different	**Nums**	- numerical constants
Fs	- formulae	**Es**	- mathematical elements $(a, +, \int, ...)$
FT	- formula token	**Rs**	- representations of a formula produced by the *generalisation algorithm*
pD	- per document	**pFT**	- per formula token
pR	- per representation	**0.**	- original formula

Table 6.1: Abbreviations

Several important characteristics of both datasets are illustrated in Table 6.2 and Table 6.3. The table includes information about the whole mirrored dataset not only about mathematical documents. Different versions of the same document have been excluded. Each formula is represented in Presentation and Content `MathML` and therefore the number of different formulae is half the number of all formulae. A formula

[3]http://cnx.org/

in the Presentation and Content format is not necessarily converted to the same internal representation and therefore they can be considered as two different formulae. It is because of the solely meaning selection technique described in Chapter 2. Both complete datasets are included in Appendix B. Although the CNX2 dataset is much smaller than the whole collection the statistical differences of presented characteristics can be neglected.

Dataset	Total Size	Size of Text Documents	Size of Mathematical Documents
`CNX1`	99.2 MB	40.19 MB	19.68 MB
`CNX2`	0.99 MB	0.79 MB	0.79 MB
Dataset	#Files	#Documents	#Mathematical Documents
`CNX1`	2454	1470	421
`CNX2`	44	16	16

Table 6.2: CNX datasets size statistics

Dataset	#Fs	A. #Fs (pD)	#Es	A. #0.Nums
`CNX1`	32306	76.736343	1539133	1.043645
`CNX2`	1416	88.500000	90680	1.492938

Table 6.3: CNX datasets overall statistics

6.2 Fine granularity

Each search engine must define its atomic search unit. Improper definition of an atom can greatly decrease the usability and speed and increase the index size. This section provides comparison of different atom definitions and their impacts on the performance. The mechanism used to split representations to atomic information unit is called *formula tokenizer*. The precise tokenizer definitions are included in Appendix B. The tokenizer looks on a formula and decides whether it is small enough to be an atomic piece of information. Otherwise it splits the formula into more parts.

There are 5 different tokenizers for CNX2 (JM1, JM2, JM3, JM4, JM5) and one for CNX1 (JM0). The definition of the CNX1 (JM0) tokenizer is the same as JM1. The statistics of the tokenizers are illustrated in Table 6.4-Table 6.6. The common used, JM1, accepts subformula up to 10 nodes with a reasonable depth difference. JM2 accepts all formulae. JM3 accepts all nodes without index nor exponent. JM4 accepts only really simple ones and the JM5 accepts only constants. JM0 is not included considered in the remainder as it operates over different dataset than the others.

Tokenizer	#FT	#Es	#D.Rs	#0.FT	#0.Es
JM0	438317	1569026	78631	79887	282717
JM1	25514	94780	3732	3869	14512
JM2	7080	95906	3732	1416	14691
JM3	7589	96267	3730	1521	14760
JM4	48811	88000	3721	7407	13949
JM5	77735	82929	3720	11969	12945

Table 6.4: Basic statistical information

Tokenizer	A. #FT (pR)	A. #Es (pFT)	A.#D.Rs	A. #0.Es (pR)
JM0	2.713533	3.579660	2.433944	3.538961
JM1	3.603672	3.714823	2.635593	3.750840
JM2	1.000000	13.546045	2.635593	10.375000
JM3	1.071893	12.685070	2.634181	9.704142
JM4	6.894209	1.802872	2.627825	1.883219
JM5	10.979520	1.066817	2.627119	1.081544

Table 6.5: Average statistical information

The corresponding graphs are illustrated in Figure 6.1 - Figure 6.4. The CNX1 dataset is not included because it is much bigger and the data obtained would distort the comparison of different tokenizers over CNX2 dataset. The first graph (#FT) shows the number of formulae. It is clearly shown that when the tokenizer accepts only constants the number of *formula tokens* is the highest. The difference between the tokenizers is signigicant, where the strictest specification contains almost 11 times the number of *formulae tokens* as the most loose one.

The number of mathematical elements illustrated in the next graph (#Es) is similar but not identical. It is caused by the algorithm producing string representation of complex *formula tokens* where e.g. the index and exponent are concatenated to the operator without a space and therefore are counted as one operand. But as it

Tokenizer	A. #FT (pD)	A. #0.Nums
JM0	1041.133017	1.057636
JM1	1594.625000	1.535311
JM2	442.500000	1.540960
JM3	474.312500	1.549435
JM4	3050.687500	1.468220
JM5	4858.437500	1.444209

Table 6.6: Additional information

can be seen it is not very common behaviour.

On the third graph (#Different Rs) in the first figure the number of different representations is illustrated. Different tokenizer specifications proved insignificant in comparison of the resulting transformations. It is also because the other parameters in the transformations are very strict and therefore allow only few mathematical operations.

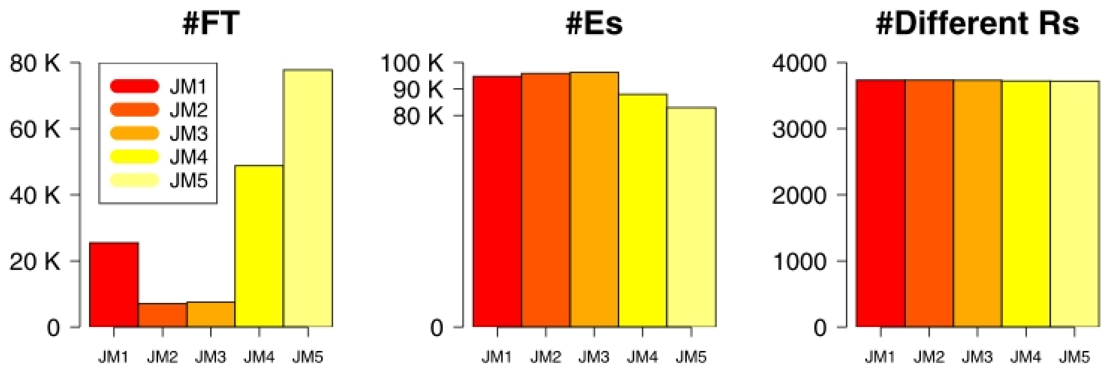

Figure 6.1: First set of characteristics

Figure 6.2 shows the same results as Figure 6.1. The difference in the number of mathematical elements across different tokenizers is up to 15%. These graphs (#0.FT, #0.Es) clearly state that both statistics, number of *formula tokens* and number of mathematical elements, are not dependent on whether only the original formula or all representations are included.

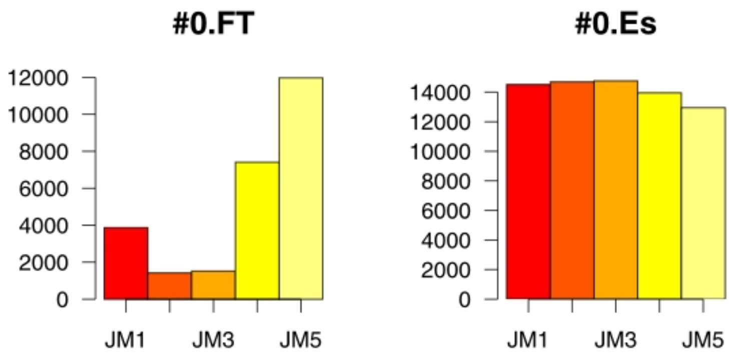

Figure 6.2: Second set of characteristics

Figure 6.3 contains the average characteristics of *formula tokens* and mathematical elements. The first two graphs (A. #FT (pD), A. #FT (pR)) show the same trend and that is the strict tokenizers having greater average counts than the less strict ones. The average number of elements (A. #Es (pFT)) is very interesting with JM2 and JM3 having almost 14 mathematical elements per *formula token*. This number is very high and almost disallows subformula search in a common formula.

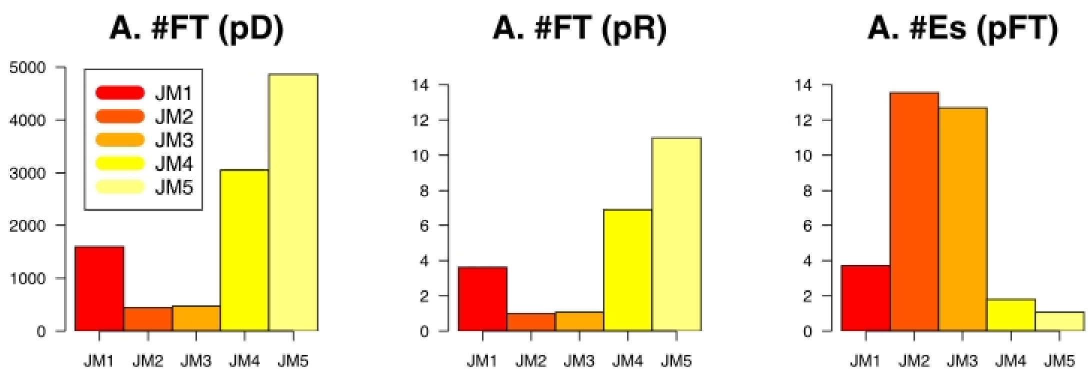

Figure 6.3: Third set of characteristics

The last characteristic figure (A. #DRs) shows that the number of different representations is almost identical among the tokenizers. It shows that approximately three out of five representations are different. When considering the properties of the transformations the first, second and third are more similar than the last ones. It influences the size of the index database where only approximately three different representations must be stored. The second graph (A. #0.Ed (pR)) tells that the other representations add two mathematical operands in average to the original formula.

The last one (A. #0.Nums) shows the average number of numerical constants in the original formulae. It indicates that proposed partial evaluation transformation is not used often because it is evaluating only numerical constants (functions like $sin, log, \ldots$ and more complex operations across different depth levels which could increase the usage are not implemented in the prototype).

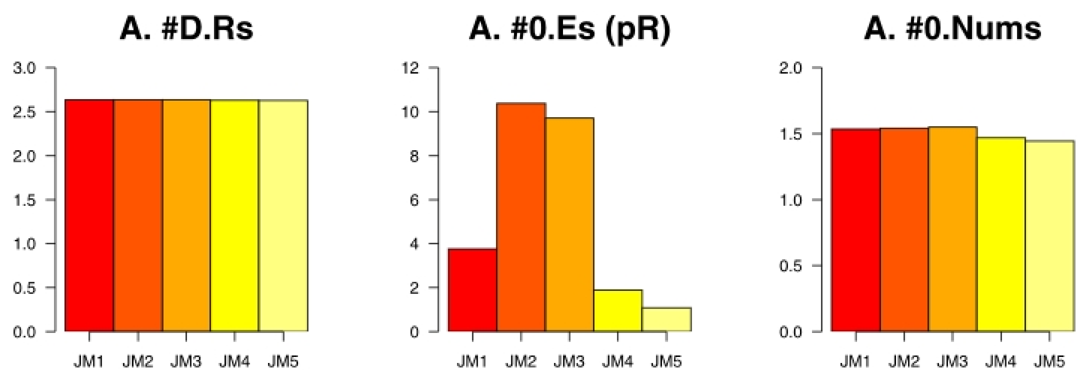

Figure 6.4: Fourth set of characteristics

6.3 Not exactly recall and precision

The recall (R) and precision (P) metrics are subjective metrics used to evaluate search engines. The definitions of recall (R) and precision (P) are:

$$P = \frac{\{relevant\,documents\} \cap \{retrieved\,documents\}}{\{retrieved\,documents\}}$$

$$R = \frac{\{relevant\,documents\} \cap \{retrieved\,documents\}}{\{relevant\,documents\}}$$

Both measures rely on a collection of documents and queries for which the relevancy of the documents is known. Every document is known to be either relevant or non-relevant to a particular query. The relevancy of a document is very subjective in mathematical searching. Many users can consider $1-cos^2(x)$ the same as $sin^2(x)$ but many can find this very disturbing as new knowledge, not necessarily known to them, is introduced. Moreover, this subjectivity spreads to whole parts of mathematics. Because there is neither official nor unofficial testing set it would be meaningless to define this set by the author itself. The creation of the set has to be based on a study of many different potential users of the mathematical search engine.

A new evaluation technique was developed which concentrates on simple mathematical formulae rather than on documents to partially eliminate the problem. The subjectivity is limited to binary decision of similarity between two formulae. The whole set is included in Appendix B and is referred to as QUERY. A sample part of the specification file is illustrated below on the left and several tested formulae are illustrated on the right. There are 82 formulae divided into original formulae and formulae which either should or should not match the original. Each formula is classified into several categories. Each category represents one characteristic transformation or feature. The categories are: COMPLEX - cpx, ORIGINAL - o, SUBFORMULA - sub, COMMUTATIVITY - com, DISTRIBUTIVITY - dis, CONSTANT ARITHMETIC - ca, CONSTANT DIFFERENCE - cd, VARIABLE ARITHMETIC - va, VARIABLE DIFFERENCE - vd and a special category ALL where all formulae are included.

```
(1/(x+2))+(3/(x-3))
CAT: o
EXP: 1
GOT: 1

1/(x+2)
CAT: sub,
EXP: 1
GOT: 1
```

```
(2*a) / (b^2+7)^{0.5} = z
s'=s*e^{ComplexI*\phi}
\sum_{i=0}^{N-1} (x_i*y_i)
F_{bot}*r = \tau
x^T*y=\sum_{i=0}^{N-1} (x_i*y_i)
IntegerS*(0*\sigma^2)
a = (a(a-7))/(a-88)
||x||_2^2=x^T*x
\tau = F*r*sin(\phi)
```

The search engine either finds or finds not the document containing the formula.

The result (RP) for one category (C) is converted in percentage terms by this formula:

$$RP = \sum_{f \in C} \frac{Query(f)}{C} \qquad \text{where } Query(f) = \begin{cases} 1 & \text{expectation matches result} \\ 0 & \text{otherwise} \end{cases}$$

where f can be included in more categories.

The QUERY was prepared by the author of the thesis and was refined by two potential users of the mathematical search engine. All formulae have been chosen from the CNX2 dataset. The selection was random but one special type of formulae has been omitted - formulae containing matrices. The reason is that: 1) it is difficult to notate a matrix, 2) the matrix must be noted exactly as no matrix transformation is very common.

The result RP metric is shown in Figure 6.5. Table on the left contains exact numerical values, the graph on the right is in percentage terms.

Dataset	QUERY
all	0.4834
o	0.6667
sub	0.5790
com	0.5000
ca	0.5000
cd	0.4000
va	0.3333
vd	0.5263
cpx	0.5455

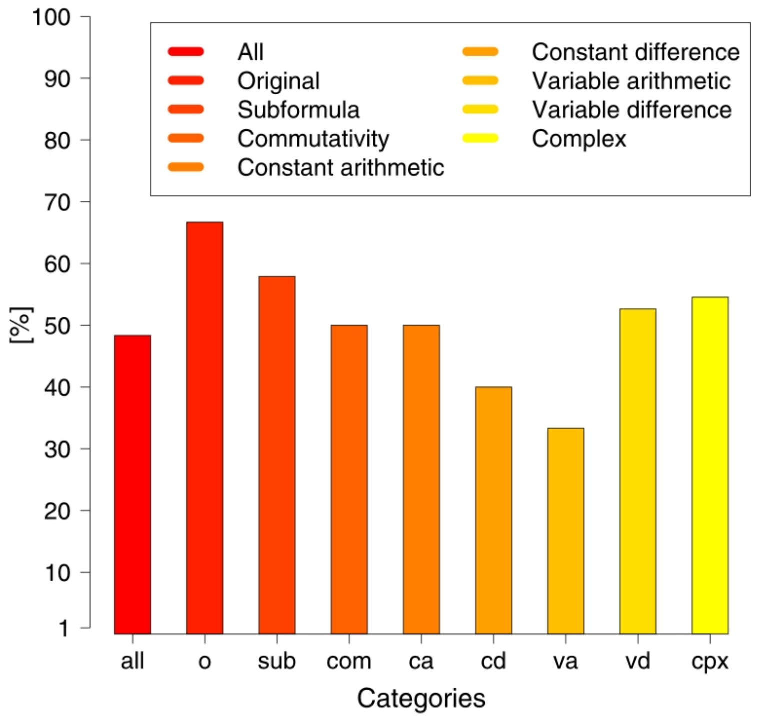

Figure 6.5: QUERY results

The Variable arithmetic test came out as the worst. It is because the variable arithmetic is not considered in the proposal. On one hand the arithmetic can prove useful but on the other hand it introduces too much new knowledge which can be confusing. The second worst test was the Constant difference. It is because the numbering of *formula tokens* is not implemented and so query $a + b$ also matches $a + a$. However this looseness happens only with the most generalised representation and can be easily excluded. The *only* 64% success of the original formula matching is caused by missing symbols and incorrect parsing of complex formulae e.g. the limits of an integral or a summation are not handled specially and in the most cases result in the unusability of the whole formula. However, this is again the implementation issue. The result of Variable and Constant arithmetic is also influeneced by the smallest number of available test cases.

It is very interesting that the Content and Presentation `MathML` format have been in most cases complementary rather than equivalent. This is caused by the lack of semantic information in the search query. The Content format kept the formula together as one word on the contrary to the Presentation format which had to introduce new mathematical operations (like $*$ when encountering ab) and therefore made the formula more complex.

All the results were influenced by two factors. Firstly, the lack of symbol definition in the implementation, the worst appeared to be the lack of commutativity by $=$, $\geq$ etc.). The second factor was that the search engine returned more results than expected. This was principally caused by the last most generalised representation.

The process of evaluation showed that the recall metric is high at the cost of lower precision. The results confirming this statement can be found in Appendix B. If this behaviour is not desirable it can be influenced either by revisiting the last transformation rule or simply by using the ranking algorithm to limit search up to specified precision.

MathWebSearch

The CNX dataset has been selected in the first place because a comparison of both engines was planned. But during the evaluation several problems have been encountered.

The exact date of the source used by *MathWebSearch* is not known, but it is estimated to be sometime around the start of 2007. Documents in CNX are versioned and *MathWebSearch* always explicitly specifies the version indexed enabling precise comparison of the same document. However, it lacks the information whether the document is indexed at all.

There are no evaluation statistics available from the *MathWebSearch* yet and therefore the evaluation had to be done manually. However, the evaluation process of *MathWebSearch* is being in progress.

The biggest problem encountered was to find more complex queries which would return any result by *MathWebSearch*. It is strongly dependant on the choice of one of three supported query languages. All three have different expression power. It has not been intuitive to find any results but for simple queries without the exact knowledge of the searching and indexing process. These problems resulted in abandoning the comparison as it would be very subjective and of almost zero information value.

6.4 Speed and size

This evaluation does not contain speed comparisons. It is clear that no existing search engine type can match full text search engine in speed. The extension must create representations of the user mathematical query which increases the time. The computation is dependent only on the input query and can be easily cached. Furthermore, the speed evaluation is meaningless without precise information about the speed and method of measurement of other search engines which is not available.

The *formula tokenizer* definitions are used from Section 6.2 and the underlying dataset is CNX2. The information about the size of the index is and the comparison is in Figure 6.6. The result was computed as the size of the whole index directory, described in Chapter 5, with text words from the documents included.

Tokenizer	Sizein Bytes
JM1	338 938
JM2	534 400
JM3	534 445
JM4	288 119
JM5	281 243

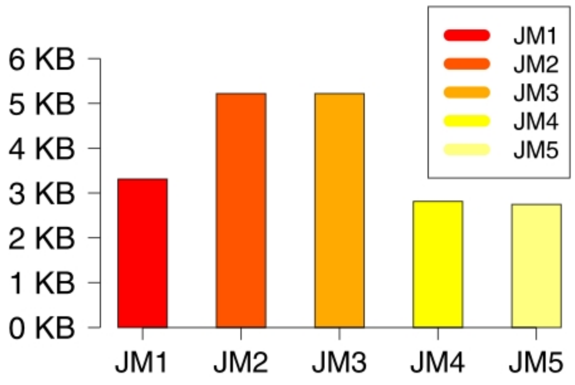

Figure 6.6: Size comparison

The JM2 and JM3 have the largest index database because they accept almost all formulae making the text word representing a *formula token* very long. This lowers the probability of existence of the same word in the database. JM4 and JM5 accept formulae only with the smallest sizes and therefore the change of already existing tokens is very high. The result of the common tokenizer, JM1, is right in the middle as could be expected.

6.5 Similarity search

One possibility how to evaluate and mathematically describe the proposed search model is to use similarity search. It is also described as very important and useful in [18]. Similarity is a well known field of interest in many research areas of computer science [24, 9].

The proposed technique is based on a full text search engine where the dynamic computing of similarity is not possible. However, it is used in the ranking algorithm to weight formulae more precisely. In the searching phase it can be used to specify minimum similarity. The last use is when designing the *formula token* specification. It can compare different specifications based on evaluation.

The definition of distance metric is taken from [18].

Definition 1. *Given a distance metric d in the "space" of mathematical expressions or patterns, and given a threshold h, two expressions or patterns E_1 and E_2 are said to be h-similar if $d(E_1, E_2) < h$.*

The distance does not need to be a distance in the topological sense. The distance metric should satisfy these properties: 1) $d(E_1, E_2) = d(E_2, E_1)$, 2) $d(E_1, E_2) = 0 \Leftrightarrow E_1 \equiv E_2$. In the second requirement the equal sign ($\equiv$) must be strictly defined because it would be very limiting in the most of the common cases . For example, if f_1, f_2 are formulae and $f_1 \equiv f_2$ means that the two formulae are mathematically equal. The second requirement would restrict the usage of this metric only to systems where $d(sin^2x, 1 - cos^2x) = 0$ holds and that is very limiting.

Assume that for an input formulae f, g the function Q generated $f_1, ..., f_N$ and $g_1, ..., g_N$. The formulae $f, f_1, ..., f_N$ are used to build a special tree with f as the root node, f_1 is the child of f, f_2 the child of f_1 etc. and analogously with g.

The distance metric d is naturally dependent on the depth of nodes in which the formulae match. Each depth level D has a weight $W(D)$. Then the metric d can be defined as the sum of weights of first nodes in which they match. First f is compared with $g, g_1, ..., g_N$, then f_1 is compared with $g, g_1, ..., g_N$ etc. till f_N. Let the first match occur at f_i and g_j. Then $d(f, g) = W(i) + W(j)$. The definition can be extended to include the structure of each formula.

Because boolean string comparison is commutative the first requirement is satisfied ($d(f, g) = W(i) + W(j) = W(j) + W(i) = d(g, f)$). The $\equiv$ operator can be defined this way: $f \equiv g$ means f and g are mathematically equal having all their operands the same but they can be permuted at the same depth level. The definition could be extended to the first depth level but than it would lack the mathematical correctness. To distinguish between matching at various depth levels the function W should satisfy: if $i \neq m, j \neq n$ and $i \neq n, j \neq m$ then $W_i + W_j \neq W_m + W_n$.

Let D be an integer, $D \in [1, 5]$ and $W : D \rightarrow [0, 100]$, then W can be defined as: $W(1) := 100, W(2) := 90, W(3) := 75, W(4) := 45, W(5) := 20$. This definition is used in the prototype.

6.6 PDF support

The implementation can also handle `pdf` document format. The `pdf` document is converted to pictures and then OCR is used. The most important conversion is from the picture format to `MathML` done by the Infty application. As the licensing policy does not allow for automatic use of the newest available version Infty version 2.4.4e is used.

One of the problems encountered is the speed of the conversion. It takes tens of seconds to convert single `pdf` file and it is very dependent on the number of pages. This drawback is notable only during the first processing of a dataset because `pdf` files are rarely updated comparing to `HTML` content. Another problem is the accuracy of the used version. The result of the comparison has not been the same for all `pdf` files. On one hand the newer version outperformed the older one in the number of recognised characters but on the other hand single characters were sometimes converted to a set of characters (e.g. M was converted to IVI). One `pdf` file could not be converted and resulted in an exception in both versions.

Small difference between the original and the converted formula can have great impact on the similarity. From the mathematical point of view also a single difference is significant and can change the meaning of a formula completely. Because of the presented drawbacks of the older version no further testing has been done. The basic test cases are included in Appendix B.

7 Summary and Discussion

Why is there no scientific search engine available? What are the problems and solutions of such a search engine? These, at first sight, simple questions have been addressed in this thesis.

In the first chapter the importance together with general problems of mathematical searching were outlined. The state of art of this research field, described in chapter two, showed that the topic is very young with few practical results available. Because there is no general theoretical background this work starts with defining the term *mathematical search engine*. The main focus was put on the applicability.

The most successful real world search engine is the full text search engine. It is the only one capable of indexing large collections of documents effectively and because of this it was selected as the base for the mathematical search engine designed in this paper.

A new theory of mathematical searching is developed in chapter three and four. To study the applicability of the theory a prototype has been implemented using Egothor v2 search engine. The prototype implementation is described in the chapter five. The important part of the new technique proposal is the evaluation described in chapter six. Again, it faces the problem of the absence of previous research. None of the analysed evaluation techniques could describe the performance of a mathematical search engine sufficiently. Therefore, not one but a set of metrics has been proposed and evaluated describing it more comprehensively. The set includes several already existing techniques or technique proposals and one new specially developed for this purpose. A prototype of a database, necessary for testing, was created and is included in Appendix B.

The engine is not primarily dependent on Content `MathML` document format or other format including semantic information. This makes the engine unique. It must be noted that common documents do not include semantic information. Moreover, it is very likely that users will not provide the query with much semantic information. This could result in a strange behaviour. For instance, `pdf` document format as the most widely used for mathematical documents can not be converted to Content `MathML` but only to Presentation `MathML`. All of these issues had to be solved.

Currently, as the only one, it can be used on a large collection of data, which is guaranteed by the underlying full text search engine. Another advantage is that the techniques developed over the last several years of the existence of publicly available full text search engine are automatically inherited and ready to use. There is no need to design, implement and test a new search engine. The extension character of the proposal allows every full text search to adopt it easily.

The search engine described in this thesis is based only on static textual representations which has been considered by many researchers as unsuitable for mathematics. But the technique has proven itself equal to already developed techniques. It is different from all other known techniques because it does not try to find a certain formula by concretising provided user query formula but on the contrary. Firstly, it tries to find an exact match and when not successful the user query formula is generalised and the search is repeated. It is definitely worth the further research efforts.

The prototype implementation of the proposed solution has already proven useful when gathering user and document statistics which are crucial for building the theory behind it. It has already offered a first glance on the real world mathematical formulae. The results show that the probability of finding a formulae is strongly dependant on the correct symbol recognition during the indexing phase. The technique of selecting one solely symbol meaning and using it during all phases solve the problem of little semantic information partially.

Another very interesting piece of knowledge obtained was that the Presentation and Content `MathML` format used to describe a mathematical formula have been complementary to each other rather than stand-alone during the evaluation.

The evaluation has illustrated problems with simple textual query language. More complex structures as matrices are very tedious to write. Practical results obtained by the evaluation have shown the exponential decrease of recall metric with the increase of the complexity of a formula. Assuming this statement holds, the increase of the recall metric by *generalisation transformation rules* is an important characteristic and the disadvantage of lower precision is therefore partially suppressed.

The fine granularity has not only influenced the usability from the user point of view but also the size of the index database. The difference is significant and therefore it must be taken into consideration.

The mathematical search engine is a very active research field. The most common problem encountered during this study was the lack of either precise definition or any previous work done. The search engine research field is very complex also without adding the necessity of defining and researching the same areas again and again. Very few attempts have been made to describe the theory or basic properties. Without widely accepted mathematical search engine characteristics, based mainly on common users rather than researchers of this field, it is very difficult to create a usable solution.

7.1 Future work

The basic research of the new technique has been already done. The future work should include the refinement of the indexing phase by introducing new transformations and analysing the overall impact on the performance, especially on the similarity. The user interface should also allow refinement of the search query to more fields than only text and mathematical.

Another main focus should be on the evaluation. It should also include the click-through[1] information from users.

In the long term horizon, the goals of the whole research field are more important than the goals of individual search engines. The main goal should be the definition of the term *mathematical search engine*. This is a non trivial task because of the different approaches taken and different user expectations. It is very likely that a universal definition is pointless and the effort should be rather on identifying the categories of mathematical search engines.

The evaluation of other available mathematical aware search engines indicate that the user must have at least basic knowledge of the underlying system e.g. the query language. Because of this the last, but not least important, is the user interface specification together with a definition of a standard query language.

Solving the above mentioned problems would simplify the development of new techniques and the research can focus on simple problems rather than on the whole research field.

[1]The act of a user clicking on a hyperlink that directs the user to another site.

Appendix A

The example transformations of several formulae are illustrated below. Each step number corresponds to a particular step in *generalisation algorithm* in Chapter 3. The examples are only one possibility of the Q transformations out of many. The reason is that the *transformation rules* are parametrised by many factors. The most important is the *formula token* specification which defines the smallest, atomic, information unit.

Example 1 (simplified cosine rule):

0. *Input formula:* $a^2 - b^2 - c^2 + 2bc\cos\alpha$
1. $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
2. **Representation:** $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
3. $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
4. $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
5. **Representation:** $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
6. $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
7. $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
8. **Representation:** $a^2 - b^2 - c^2 + 2\cos(\alpha) * b * c$
9. $a^2 - b^2 - c^2 + const\ \cos(\alpha) * b * c$
10. $a^2 - b^2 - c^2 + const\ \cos(\alpha) * b * c$
11. **Representation:** $a^2 - b^2 - c^2 + const\ \cos(\alpha) * b * c$
12. $id_1^2 - id_2^2 - id_3^2 + const\ \cos(id_4) * id_2 * id_3$
13. $id_1^2 - id_2^2 - id_3^2 + const\ \cos(id_4) * id_2 * id_3$
14. **Representation:** $id_1^2 - id_2^2 - id_3^2 + const\ \cos(id_4) * id_2 * id_3$

Example 2:

0. *Input formula:* $2\pi(a^2 + \frac{b^2}{2\varepsilon}\ln\frac{1+\varepsilon}{1-\varepsilon})$
1. $2\pi(a^2 + \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon})$

2. **Representation:** $2\pi(a^2 + \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon})$
3. $2\pi(a^2 + \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon})$
4. $2\pi(a^2 + \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon})$
5. **Representation:** $2\pi(a^2 + \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon})$
6. $2\pi * a^2 + 2\pi * \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon}$
7. $2\pi * a^2 + 2\pi * \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon}$
8. **Representation:** $2\pi * a^2 + 2\pi * \ln\frac{1+\varepsilon}{1-\varepsilon} * \frac{b^2}{2\varepsilon}$
9. $const * \pi * a^2 + const * \pi * \ln\frac{const+\varepsilon}{const-\varepsilon} * \frac{b^2}{const*\varepsilon}$
10. $const * \pi * a^2 + const * \pi * \ln\frac{const+\varepsilon}{const-\varepsilon} * \frac{b^2}{const*\varepsilon}$
11. **Representation:** $const * \pi * a^2 + const * \pi * \ln\frac{const+\varepsilon}{const-\varepsilon} * \frac{b^2}{const*\varepsilon}$
12. $const * \pi * id_1^2 + const * \pi * \ln\frac{const+id_1}{const-id_1} * \frac{id_1^2}{const*id_2}$
13. $const * \pi * id_1^2 + const * \pi * \ln\frac{const+id_1}{const-id_1} * \frac{id_1^2}{const*id_2}$
14. **Representation:** $const * \pi * id_1^2 + const * \pi * \ln\frac{const+id_1}{const-id_1} * \frac{id_1^2}{const*id_2}$

Appendix B

One part of the master thesis is the evaluation of the proposed technique. The implementation of the technique together with all utilities and datasets used for evaluating purposes are included in the attached CD.

The distribution is focused on the extended Egothor v2 search engine. Additional libraries and applications, with sources and different versions when available, are included too. The documentation includes a draft version of a description of Egothor v2. Evaluation tools developed especially for this master thesis are included as subprojects. Other statistics have been generated using scripts spread over the distribution. The evaluation dataset collection includes some `pdf` files and their `MathML` representations, documents from `cnx.org` and other documents used also for testing.

The general description of the CD content is in the main folder in `readme` file.

Appendix C

The following listings demonstrate several important design paradigms of the implementation. The whole implementation can be found in Appendix B.

The paradigm of filtering streams are presented in listings C.1-C.2. The trimmed output of the symbol mapping file is in Listing C.3. The definition of one *generalisation algorithm* is illustrated in Listing C.4.

Both specification files are included in Appendix B:

- symbol specification - in `settings/math/symbols.xml`
- algorithm specification - in `settings/math/algorithms.xml`

```
final public void push (IElementFilter<E> tf) {

  if (null == _firstf)
    // first call to push
    _firstf = tf;
  else
    // append next filter
    _lastf.filter (tf);

  // save last filter
  _lastf = tf;
}
```

```
final public void push (IElementInDevice<E> td) {

  if (null == _firstf)
    // no filters
    _firstf = td;
  else
    // append next filter
    _lastf.filter (td);

  _device = td;
}
```

Listing C.1: Implementation of `push()`

```
public Token read (IElementRead<Token> nextf) {

  // Anything to read
  if (!_empty_device) {
    Token token = nextf.read ();
    if (null != token)
      return token;
    _empty_device = true;
  }

  // Return math tokens or null if empty
  return _tokens.pollFirst ();
}
```

Listing C.2: Implemenation of `read()`

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
<!--     CONSTANTS -->
 <constant representation="ExpE">
  <alternative>expe</alternative>
  <alternative>exp</alternative>
  <alternative>ExponentialE</alternative>
  <alternative>e</alternative>
  <alternative>ee</alternative>
  <alternative>âĚĞ</alternative>
 </constant>
...
<!-- VARIABLE -->
 <variable representation="Gamma">
  <alternative>gamma</alternative>
  <alternative>ÎŞ</alternative>
  <alternative>Îş</alternative>
 </variable>
...
<!--   OPERATORS -->
 <operator representation="(" priority="HIGHEST" arity="1">
  <alternative>{</alternative>
  <alternative>[</alternative>
 </operator>
...
<!-- UNARY -->
 <unary representation="!-" />
 <unary representation="Forall">
  <alternative>forall</alternative>
  <alternative>âĹĂ</alternative>
 </unary>
...
<!-- BINARY -->
 <binary representation="NotEq" priority="MED">
  <alternative>noteq</alternative>
  <alternative>neq</alternative>
  <alternative>âĽ </alternative>
 </binary>
...
</definition>
```

Listing C.3: Symbol mapping file

```
<?xml version="1.0" encoding="UTF-8"?>

<specification>

 <algorithm name="basic">

  <transformation name="order" />
  <output priority="1" />

  <transformation name="eval" />
  <transformation name="approximate" />
  <transformation name="order" />
  <output priority="2" />


  <transformation name="distributivity" />
  <transformation name="multiplying" />
  <transformation name="owndenominator" />
  <transformation name="order" />
  <output priority="3" />

  <transformation name="constants2const" />
  <transformation name="order" />
  <output priority="4" />

  <transformation name="unknown2id" />
  <transformation name="order" />
  <output priority="5" />

 </algorithm>


</specification>
```

Listing C.4: Definition of one *generalisation algorithm*

Bibliography

[1] Sergey Brin and Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine,
http://infolab.stanford.edu/∼backrub/google.html

[2] Amit Singhal: Challenges in Running a Commercial Web Search Engine,
http://singhal.info/SIGIR-Keynote.pdf

[3] Dirk Draheim, Winfried Neun, and Dima Suliman: Classifying Differential Equations on the Web, Springer-Verlag Berlin Heidelberg 2004.

[4] Michael K. Bergman: The Deep Web: Surfacing Hidden Value, July 2000, BrightPlanet.com LLC.

[5] Robert Miner: Enhancing the Searching of Mathematics,
http://www.ima.umn.edu/complex/spring/math-searching.html

[6] Sun B., Tan Q., Mitra P. and Giles C.L.:Extraction and Search of Chemical Formulae in Text Documents on the Web, The 16th International World Wide Web Conference, (WWW'07).

[7] Michael Yang, Richard J. Fateman: Extracting mathematical expressions from postscript documents, ISSAC 2004, p.305-311.

[8] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida and T. Kanahori: INFTY - An integrated OCR system for mathematical documents, Proceedings of DocEng 2003, Grenoble, France.

[9] Pang-Ning Tan, Michael Steinbach and Vipin Kumar: Introduction to Data Mining, Addison-Wesley, May 2005.

[10] Mitchell, T.: Machine Learning, McGraw Hill, 1997.

[11] Jürgen Stuber, Mark van den Brand. Extracting Mathematical Semantics from LaTeX Documents, PPSWR 2003, LNCS 2901, pp. 160-173, Springer, 2003.

[12] Galamboš L., Sherlock W.: Getting Started with ::egothor, Practical Usage and Theory Behind Java Search Engine, 2004,
http://www.egothor.org/

[13] Mathematical Markup Language (MathML) Version 2.0,
http://www.w3.org/TR/2003/REC-MathML2-20031021/

[14] Krzysztof Retel and Anna Zalewska: Mizar as a Tool for Teaching Mathematics,
http://merak.pb.bialystok.pl/mkm2004/KRetelAZalewska.pdf

[15] Adobe Systems Incorporated, Postscript Language Reference, Third Edition. Reading, Massachusetts: Addison-Wesley Publishing Company, 1999.

[16] Adobe Systems Incorporated, PDF Reference, Sixth Edition,
http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference.pdf

[17] Nikita Synytskyy, James R. Cordy, Thomas R. Dean: Robust Multilingual Parsing Using Island Grammars,
http://www.cs.queensu.ca/∼cordy/Papers/CASCON03_MultiParsing.pdf

[18] A. Youssef: Roles of Math Search in Mathematics, Proceedings of the 5th International Conference on Mathematical Knowledge Management, Lecture Notes in Computer Science 4108, Springer-Verlag, 2006.

[19] Michael Kohlhase and Ioan Sÿucan. A search engine for mathematical formulae, In Ida et al. [ICW06], p.241-253, 2006.

[20] Serge Bondar: Search Engine Indexing Limits: Where Do the Bots Stop?,
http://www.sitepoint.com/article/indexing-limits-where-bots-stop

[21] F. Guidi: Searching and Retrieving in Content-based Repositories of Formal Mathematical Knowledge. Ph.D. Thesis in Computer Science, University of Bologna, March 2003. Technical report UBLCS 2003-06.

[22] Bruce R. Miller, Abdou Youssef: Technical Aspects of the Digital Library of Mathematical Functions. Ann. Math. Artif. Intell., p.121-136, 2003.

[23] A. Riazanov and A. Voronkov: Vampire 1.1 (system description). In R. Goré, A. Leitsch, and T. Nipkow, editors, Automated Reasoning - First International Joint Conference, IJCAR 2001, LNAI 2083, Springer, 2001.

[24] Dr. Horst Eidenberger: Visual Information Retrieval, Wien, 2004.

[25] Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv: XSEarch: A Semantic Search Engine for XML, Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[26] Sihem Amer-Yahia, Chavdar Botev, Jochen Dörre, Jayavel Shanmugasundaram: XQuery Full-Text extensions.

[27] The MKM Interest Group:
http://www.mkm-ig.org/

[28] Andrea Asperti, Bruno Buchberger, James H. Davenport (Eds.): Mathematical Knowledge Management, Second International Conference, MKM 2003, Bertinoro, Italy, February 16-18, 2003.

[29] Andrea Asperti, Grzegorz Bancerek, Andrzej Trybulec (Eds.): Mathematical Knowledge Management, Third International Conference, MKM 2004, Bialowieza, Poland, September 19-21, 2004.

[30] Michael Kohlhase (Ed.): Mathematical Knowledge Management, 4th International Conference, MKM 2005, Bremen, Germany, July 15-17, 2005.

[31] Jonathan M. Borwein, William M. Farmer (Eds.): Mathematical Knowledge Management, 5th International Conference, MKM 2006, Wokingham, UK, August 11-12, 2006.

[32] The ACM Digital Library,
http://portal.acm.org/guide.cfm

[33] K. Maly, M. Zubair, M. Nelson, X. Liu, H. Anan, J. Gao, J. Tang and Y. Zhao: Archon - A Digital Library that Federates Physics Collections, DC 2002, Florence, October 2002.

[34] CiteSeer: Scientific Literature Digital Library,
http://citeseer.ist.psu.edu/

[35] GalambošL.: EGOTHOR,
http://www.egothor.org/docs/e2.pdf

[36] The LogiCal project,
http://logical.inria.fr/

[37] Math WebSearch (A semantic search engine),
http://search.mathweb.org/

[38] Mathdex search engine,
http://www.mathdex.com:8080/mathdex/search

[39] Paul Libbrecht, Erica Melis: Semantic Search in LeActiveMath,
http://www.activemath.org/

[40] N. J. A. Sloane: On-Line Encyclopedia of Integer Sequences,
http://www.ams.org/notices/200308/comm-sloane.pdf

[41] SpringerLink,
http://www.springerlink.com/home/main.mpx

[42] PDFedit program,
http://sourceforge.org/projects/pdfedit

[43] Formal Methods Group at University of Manchester, http://www.manchester.ac.uk/research/areas/subareas/?a=s&id=54592

[44] Research Group Automated Reasoning at Technische Universität München, http://www4.informatik.tu-muenchen.de/ schulz/WORK/eprover.html

[45] Kurt Bryan, Tanya Leise: The $25,000,000,000 eigenvector. The linear algebra behind Google, SIAM Review, 48 (3), 2006.