Natural Language and Mathematics Processing for Applicable Theorem Search

by

Ștefan Anca

a thesis for conferral of a Master of Science in Computer Science

Prof. Dr. Michael Kohlhase
(Jacobs University)

Prof. Dr. John Bateman
(Universität Bremen)

Date of Submission: August 24th, 2009

School of Engineering and Science

Declaration

The research subsumed in this thesis has been conducted under the supervision of Prof. Dr. Michael Kohlhase from Jacobs University Bremen. All material presented in this Master Thesis is my own, unless specifically stated.

I, Ştefan Anca, hereby declare, that, to the best of my knowledge, the research presented in this Master Thesis contains original and independent results, and it has not been submitted elsewhere for the conferral of a degree.

Ştefan Anca
Bremen, August 24th, 2009

Acknowledgements

I would like to first of all thank my supervisor, Prof. Dr. Michael Kohlhase, for his continuous guidance throughout my entire research within the KWARC group and specifically for this thesis. Using his advice and direction, I have managed to fully grasp the complexity of my research topic and develop my own approach to pursue it. His comments on the final form of this thesis were most valuable and led me to a comprehensive exposition.

I would also like to thank the members of the KWARC group for maintaining an exciting research environment, encouraging independent study and research.

I also extend my gratitude to the developers of MathWebSearch for continuous support in my research efforts.

I would like to thank my friends who took the time to review drafts of this report, especially Andrei Giurgiu, Adrian Djokic, Magdalena Narozniak and Milena Makaveeva.

Ultimately, I would like to extend a warm thank you to Zhenya Samoilova, for proof-reading but especially for continuous encouragement and emotional support during my most stressful times.

Abstract

This thesis tackles the problem of information extraction from scientific documents which contain combined natural language and mathematical discourse. More specifically, the project described in this thesis focuses on extracting *idioms*, fixed-structure natural language formulations containing both text and mathematics from XML documents, for the purpose of providing an applicable theorem retrieval service for the user. The goal of this thesis is to provide a search engine utility that helps scientists in their research. By searching for theorem-like formulations and indexing their mathematical content, we can retrieve applicable theorems to user queries through generalization search. This thesis introduces two systems to achieve its goal. First, an idiom extraction tool called IDIOM SPOTTER employs pattern-matching techniques to retrieve idioms from scientific texts. Then, the APPLICABLE THEOREM SEARCH system combines the idiom extraction of the IDIOM SPOTTER with the mathematical term search of the MATHWEBSEARCH system to retrieve applicable theorems. By analyzing the frequency of occurrence of idioms in large databases of documents, we draw conclusions about the scientific and linguistic orientation of the documents and common linguistic practices within different fields. This thesis brings together semantic extraction techniques from mathematics and natural language for the purpose of understanding the combined semantics of informal natural language and mathematical discourse.

Contents

List of Figures

Chapter 1

Introduction

The World Wide Web has become a huge database of scientific information, with more and more academic articles being published in an online format. For scientists in all fields of study, this information would be a great source of knowledge, if it was easily accessible through a semantic service. At present time, conventional text search engines provide limited support for retrieving more meaningful information than string comparison-based search results, which have nothing to do with the meaning of the matched text. We see the need for new types of search services targeted at scientists, semantic services which would provide more context information and a deeper meaning of the returned results.

This thesis proposes a method to help mathematicians and scientists find applicable theorems to their objects of research. Having an instance of a mathematical object which can be manipulated in various ways is useful only when we know how to bring it forward. Without knowing which mathematical theory would fit our instance and how to apply it in our particular case, we are stuck. This scenario can occur quite often, especially in the case of natural scientists whose objects of research frequently need advanced mathematical support. The project described in this thesis, entitled APPLICABLE THEOREM SEARCH, aims at telling scientists exactly which theorems can be applied to their particular mathematical formula and how they can further manipulate it, based on the knowledge and information that other scientists have contributed to the Web database of scientific documents. The "Web database" is not a particular database, but all the collections of scientific documents which can be indexed and aggregated into one generic database, representing the scientific information available on the Web.

The analytical task that this thesis sets forth to achieve is to extract fixed format natural language formulations, containing math, called *idioms*, from scientific publications. Then, by indexing the mathematical formulas in these idioms according to their predefined structure, the system will offer a search interface for applicable theorems.

This introductory chapter serves to briefly describe the object of research of this thesis to the reader. In the following sections, I will help the reader quickly grasp the problem and my approach at solving it by first providing a motivation for and a running example of the desired end system. Then I will set forth the goals of this research project and explain the steps taken to achieve them.

1.1 Motivation

An online mathematical tool in today's world should help the scientist gain access to information more easily, help him/her sift through the vast knowledge base that is the Internet. Oftentimes, the information that a scientist needs is out there somewhere, but hard to find, in an intricate World Wide Web, which measures somewhere around 20 Billion webpages [web]. Finding information by keyword search is the norm today, but one needs to enter increasingly long strings of text in order to narrow down the result list to a reasonable size. Most often, users rarely click past the first 3 pages of results returned by a Google [Goo09] query. But even with the tens or hundreds of thousands of hits that a search for common words will return, a string search will never retrieve applied knowledge or semantic information for the user. For example, a search for "*European blue car with 2 doors and a CD player*" in any search engine will not return a list of all car models with 2 doors, which come in different shades of blue, have an in-built CD player and are manufactured in Europe. Similarly, when a scientist wants to search for the result of the Schrödinger Wave Equation applied to a a spin-1/2 wave function $\overrightarrow{\Psi} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$, the user is at a loss.

Searching for text in natural language documents has always been a great tool for researchers in all fields of study. At present time, text in a great variety of formats, ranging from clear text to PDF or PPT is indexed by search engines such as Google. However, in the world of natural sciences, mathematicians, physicists and computer scientists alike are missing customized search tools, more appropriate for their needs. Searching for formulas, definitions or theorems is essential to the natural sciences and not only, and therefore, scientists can considerably benefit from using such a service, since it would render them access to information that follows some strict patterns of representation.

The Applicable Theorem Search project presented in this thesis aims at providing a large theorem database which can be queried through mathematical terms, a resource which could be used by scientists in all fields all around the world. Being able to query for applicable theorems to a term of interest is equivalent to having an all-knowledgeable theorem assistant.

1.2 A running example

In order to provide a practical motivation for this thesis, the following running example will be given and used to guide the description of the various parts of the project. The current thesis proposes a system that will offer scientists access to generalized knowledge. In many cases, scientists are not aware that their specific research problems represent instances of larger, more general problems, which have already been solved.

Let us assume the illustrative example[1] of a cell biologist, investigating the effects of cancer on human cell decay. After some empirical research in the lab, she has deduced a recursive formula for the cell decay rate, as influenced by the disease:

$$C(t) = 9C\left(\frac{t}{3}\right) + r(t)$$

where $r(t)$ is the linear old rate. This formula tells her that the disease influences the growth rate to some extent, but she needs to solve the recursive equation in order to find out the new rate of decay. Unfortunately, she never studied complexity theory and she does not know that she is looking at a classic case of the Master Theorem [CLRS01]:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a = 9$ and $b = 3$.

Thus, she will open the web page of the Applicable Theorem Search system [Mat] and enter her recursive equation. The system will do generalization search with the concrete values inputted in the equation and retrieve an *if - then* theorem construction, namely the first case of the Master Theorem:

Case 1: If given a recurrence relation of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

Figure 1.1: Running Example: Master Theorem

With the help of this theorem, the scientist quickly determines the asymptotic bound for the new cell decay rate, which is $C(t) = \Theta(n^2)$.

[1]This example is conceived purely for expository reasons and should not be taken literally, as it has no biological research foundation.

As can be seen, our scientist's problem is an instance of a classical Computer Science recurrence problem, but which in Biology might be nonstandard. The Applicable Theorem Search system takes the instance of her problem, generalizes the equation and finds a match with a theorem in the idiom database, which was retrieved and displayed to the user (the concept of an *idiom* will be fully explained in Chapter 3). Thus, a scientific research problem can be solved, based on the knowledge that previous scientists contributed to the web databases, knowledge which was semantically "understood" and indexed by a search engine so that it can be retrieved by user queries later.

1.3 Research Goals

The broad research goal of this thesis is to perform combined natural language and mathematics analysis for information extraction. By looking at text combining mathematics and natural language, I plan to extract semantic information which falls into predefined patterns. Indexing the mathematical part of the extracted information enables special search functionality which directs the user from specific terms to general mathematical expressions at search time.

More specifically, the main goal of my thesis project is to provide a tool for *applicable theorem search* on large scientific corpora. The corpora that will be used as a basis for experiments are the Connexions corpus [CNX09] and the arXMLiv [arX09] database of documents, transformed from LaTeX into an XML format. The scientific texts contained in these corpora are analyzed and the interesting parts are indexed. What is meant by the interesting parts within the scope of this project is natural language constructs called *idioms*, specific to the rhetoric style of these texts. A system called Applicable Theorem Search is employed for indexing these texts. The search engine capabilities are used to match generalization queries and produce applicable theorems.

The most challenging and interesting research objective is the natural language processing task which involves spotting the idioms and giving them some meaning (semantics). This task requires understanding the morphological structure of such constructs, the type of words authors usually use and how to mix in mathematical formulas with natural language. The objective here is to be able to spot idioms by just matching structural patterns to the text, and also to understand the semantics behind them. This is a question of combining natural language and mathematics and extracting some meaning from the text. The practical application implementing the extraction process is the Idiom Spotter system.

Another objective is mixed natural language and mathematics indexing and querying for idiom structures. We build upon the MathWebSearch system, which can index mathematical formulas. Thus, one of the goals of the thesis is to provide an extension to the already-existing MathWebSearch system so that it can store entire chunks of text and make them available for search. The end service should be presented in a nice graphical presentation to the user. The graphical user interface of the proposed Applicable Theorem Search system should provide services similar to the current MathWebSearch system [ATS].

The practical goal of this thesis is to provide the working Applicable Theorem Search system. This system should function as a regular search engine, crawling large corpora of scientific documents (containing theorems), extracting all the theorem excerpts from within and indexing them based on their mathematical formulas. Then, at search time, the Applicable Theorem Search system should take a mathematical term as a query and retrieve all theorems which can be applied to this term from its databases. The Applicable Theorem Search system is built as a combination between the Idiom Spotter, which retrieves the theorems and MathWebSearch, which indexes the mathematical terms in them. Thus, these two components need to be created (in the case of the Idiom Spotter) or thoroughly analyzed (in the case of MathWebSearch) before they can be merged together.

1.4 Structure

In the following chapters, this thesis will be structured as follows: First, I will provide a short review of related work and the status quo in the field of natural language and mathematics processing in Chapter 2. Then, I will continue with an introduction into the field of Natural Language Processing, from the point of view of *idiom spotting* in XML documents. This will be presented in Chapter 3, alongside the theory behind mathematical term search. Chapter 4 will bring a description of the Idiom Spotter system, the implementation of the theoretical idiom retrieval concepts mentioned in Chapter 3. Then, I will provide an in-depth analysis of the architecture of the MathWebSearch system, in Chapter 5. A very important part of this chapter will describe and evaluate the design decisions taken and methods attempted in the implementation of the system extension towards Applicable Theorem Search. Chapter 6 of this report will describe the Applicable Theorem Search system, the embodiment of the entire theoretical framework and an application that retrieves theorem-like idioms to the user. It will also provide an analysis of the main challenges faced by the system. In Chapter 7, I will provide a description of the causes of the main problem encountered by the Applicable Theorem Search and a possible solution for it. In the final Chapter 8, I will evaluate the work subsumed in this thesis and provide an outlook for future work and development that can be contributed to the topic. The last part of this document will consist of a conclusion.

Chapter 2

State of the Art and Related Work

This chapter will present a short review of the most relevant current research projects focused on natural language and mathematics processing from (or to) XML documents. We will look at mathematics and language transformation, formal mathematics representation, mathematics search and knowledge extraction from XML documents. This chapter is divided into three parts: The first part corresponds to the research done within (or in collaboration to) the KWARC research group at Jacobs University, while the second part corresponds to projects pursued in external research centers. The reason for this separation is my familiarity with the projects inside the KWARC research group, that I am a member of. The third part provides a short discussion of the research performed so far on the topic that the Applicable Theorem Search system tackles and the improvements that I plan to bring through this system.

2.1 KWARC

This project is part of the research conducted within the Knowledge Adaptation and Reasoning for Content (KWARC) [KWA09] group, led by prof. Michael Kohlhase at Jacobs University. The main research focus of our group is knowledge representation with a view towards applications in knowledge management. KWARC tries to extend techniques from formal methods to be used in settings where formalization is either infeasible or too costly. We concentrate on developing techniques for marking up the structural semantics in technical documents.

2.1.1 MathWebSearch

Within its project activities, the KWARC group has focused extensively on the representation and the search for mathematics. At the moment, the group benefits from the

MathWebSearch(MWS) [Mat09] system, a search engine that can index mathematics found in content representation on the web. First introduced in [KS06], MathWebSearch is a system which employs substitution tree indexing and provides specific math querying capabilities. Currently indexing the Connexions [CNX09] repository of scientific documents and the Wolfram repository of mathematical functions [Wol09] (both containing *Content MathML* formulas), as of its last version, MathWebSearch can retrieve documents based on user generated XML queries or math formula queries generated by the interactive JavaScript-based *Sentido* interface [KAJ$^+$08]. Version 0.4 also benefits from new search capabilities like generalization, variation and unification queries. MathWebSearch is a very useful tool in the information extraction enterprise, because it can understand and index mathematics in the Content MathML format. Additions like the *MaTeSearch* combined text+math search capabilities [Anc07] have been made to enhance the power of the system and to improve its features. However, the biggest problem that MathWebSearch is faced with at current time is the lack of scientific documents which represent their mathematics in a semantic format. An extensive description of the MathWebSearch functioning and architecture is given in Chapter 5.

2.1.2 arXMLiv

One of the largest-scale projects of the KWARC group, the arXMLiv [arX09] project is an enterprise which tries to transform the large *arXiv* [arX07] collection of scientific documents into a formal XML representation, for easier machine processing [SK08]. The database, administered by Cornell University, is a collection of more than 500000 documents, written in usual LaTeX source formatting. The arXMLiv project employs the external LaTeXML [Mil09] tool, a program that converts LaTeX documents into XHTML format. Because, as the authors claim [Mil09], LaTeXML tries to mimic TeX's behavior, but to output XML instead of DVI, native support for each LaTeX package needs to be implemented. Since there are literally thousands of LaTeX packages available at the moment, and scientists like to pick different combinations of them when writing papers, the original LaTeXML system was built with support for only a few selected ones, enough to transform the documents in the Digital Library of Mathematical Functions(DLMF) [DLM05]. The DLMF is a small database of documents written with a restricted set of LaTeX packages and can be perfectly transformed to XML by this tool. However, the documents in the *arXiv* come from various fields of study like astro-physics, mathematics and computer science and need more bindings in order to be properly transformed to XML. One of the main objectives of the arXMLiv project is to provide more bindings for this translation, in order to enrich the power of LaTeXML. Because of the large size of the corpus and the need for recurring runs on it, arXMLiv employs a build system consisting mainly of a distributed file system, a queue manager, a build manager and `make` jobs on each of the hosts [SK08]. This corpus is one of the most valuable resources of the KWARC research group and it provides the perfect "playground" for the development of various tools, dealing with information extraction or annotation on large corpora, like the current proposed Applicable Theorem Search system.

2.1.3 LaTeXML

The LaTeXML converter is also interesting from the point of view of this thesis when looking at the mathematical formulas it translates. The LaTeXML system was built by Bruce Miller and is not part of research inside the KWARC group. However, since its creator is a close collaborator of the arXMLiv project, I am mentioning it here as KWARC-related work. LaTeX is a presentation-oriented format for the representation of mathematical formulas. The most useful conversion from it to XML for us would output a semantic-oriented format like *Content MathML*. At the moment, the postprocessing part of LaTeXML can flawlessly do a good transformation to *Presentation MathML* (which is still ambiguous in terms of the meaning of the mathematical symbols) and a transformation which makes a lot of wrong assumptions to *Content MathML*. But current research efforts are being made in the KWARC group and the specific LaMaPUn project to improve this latter transformation and hopefully reach a correct final Content MathML.

2.1.4 LaMaPUn

The *Language and Mathematics Processing and Understanding* (LaMaPUn) project was started by small group of graduate students, under the guidance of Michael Kohlhase inside the larger KWARC research group. The LaMaPUn research aims at the development of combined mathematics and natural language processing techniques on the arXMLiv corpus. The long term goals are the development of tools like Part-of-speech taggers, parsers and semantical analyzers for XML documents with mathematical content. The current LaMaPUn architecture provides a workflow and semantic tools for the correct transformation of the mathematics in the *arXiv* into Content MathML. For more details, the architecture and its usefulness to the Applicable Theorem Search system is described in Section 7.3.

2.2 External - Related Work

Related research topics from computer scientists all around the world give a lot of insight into many of the problems that the current thesis project might be faced with, as it develops. It is useful to get to know these problems and learn from the experience of those who have studied and have dealt with them before by analyzing external related work.

2.2.1 Formal Mathematical Language

Mathematics is a formal language. But mathematical proofs are not. In any scientific document, authors freely use a combination of natural language discourse and mathematical formulas to enounce or prove a statement. In order to verify or extract any concepts from such texts, the language needs to be parsed and "understood" by computers. The approach of [VLPA08] to this is to create a strict formal language, *ForTheL*, to replace free discourse of mathematics. ForTheL is defined in cleartext format but simple and restricted enough to be used as an easy input for the System for Automated Deduction(*SAD*) proof assistant, just like program code [PVLA07]. Thus, famous mathematical theories, written before computers were in existence can be encoded in a formal representation and automatically and rigorously verified for correctness. As the authors claim, the translation process from mathematical text to *ForTheL* must be done manually, so that the output is semantically correct and free of the inherent ambiguities of a natural language based on context, scope and presentation [VLPA08]. This method of mathematical text processing requires a strict grammar and direct translation to the formal language of any processed mathematics.

2.2.2 Informal Mathematical Discourse

Claus Zinn's PhD thesis on "Understanding Informal Mathematical Discourse" takes a different approach at proof-checking, by trying to parse and understand the mathematical proofs in their informal presentation. In his exposition [Zin04], C. Zinn gives a thorough linguistic and mathematical analysis of mathematical proofs and all constructions that can appear inside these mathematical objects. Building up an extension of Discourse Representation Theory [Kam95], he then proposes a computational framework for understanding informal mathematical discourse using Proof Representation Structures, based on proof lines. A discourse update engine incorporates the underspecified semantic representation of single proof lines into the proof context of a PRS by making use a proof planner [Zin04]. The prototype concept implementation of this theory is the '*'Verifying Informal Proofs*" system (`Vip`).

2.2.3 MathML

Another way to formally specify mathematics is, of course, the *MathML* XML format, which can encode both the presentation and the semantic form of mathematical formulas. The presentation form, *Presentation MathML*, is row-oriented format, focused on visual representation. The semantic form, called *Content MathML* is, as the name suggests, oriented towards content and the underlying semantics of the formula, thus posing no ambiguity related to the formal mathematical meaning. But most mathematics in scientific papers does not come in *Content MathML* format. Authors usually write the math in visual editors or in presentation format, leaving little hints of the actual explicit semantics of their formulas, apart from the document context. In order to reach the strict unambiguous format, translation tools must be applied, to bring the formulas from their original representation to something formal and explicit. Since most of the authors write their scientific works in LaTeX , a lot of these translators work directly on the LaTeX sources. Some examples of these tools are Bruce Miller's LaTeXML [Mil09] and Richard Fateman and Eylon Caspi's TeX to mathematics parser. The latter authors present a lot of the difficulties in deciphering the ambiguous representation that TeX inherently outputs, because of its typesetting orientation [FC99]. These difficulties, arising mostly from the different interpretations of mathematical symbols in complex formulas are the main issues tackled by mathematical formula disambiguation, which needs to be done in order to reach a semantic computer parseable format.

2.2.4 SciBorg

Also researching the extraction of specific scientific information from documents, the *SciBorg* project [Sci08] tries to combine several existing tools for a unified analysis, using one basic information format. Focusing several Natural Language Processing (NLP) tools on working with Robust Minimal Recursion Semantics (RMRS), the project attempts the extraction of information from scientific papers in the field of chemistry [CCMr$^+$06]. The project appeals to classical NLP tools, like tokenizers, part-of-speech (POS) taggers and parsers for various analyses of the same text. The system architecture creates both shallow and deep analyses of the text in RMRS representation in order to gain more information from the content of documents. SciBorg aims to develop a discourse analysis of scientific papers based on the rhetorical role of citations, the determination of scientific attribution for specific intellectual content and Argumentative Zoning [RCC$^+$08]. RMRS [HC06] is chosen as the universal representation format for all these NLP tools because of its compatibility with external applications which can run on the RMRS output. RMRS can be used with shallow (but faster) processing techniques, such as POS taggers, noun phrase chunkers and stochastic parsers, which don't need detailed lexicons. At the same time, deep RMRS analyses can be produced in situations where deep parsing can be applied. Where there is not enough processing capability or lexicon capability, then the shallow analysis is default. Both the deep PET/ERG parser and the shallow Robust Accurate Statistical Parsing (RASP) system are run on chemistry documents and their output is brought to RMRS format alongside the output of the OSCAR-3 [CMR06] tool for shallow, chemistry-specific named entity parsing. The RASP system is the backbone of the SciBorg processing architecture, performing sentence splitting, tokenizing, parsing and POS tagging. Then, the resulting RMRS analyses are matched, merged and munged [Cop07] in robust operations aiming at information extraction, semi-automatic ontology construction in OWL and Research markup (Argumentative Zoning annotation). This output is then further enriched by anaphora resolution and word sense disambiguation (WSD) to create the final annotations [CTW06].

2.2.5 VSDITLU

The Applicable Theorem Search project aims to provide functionality that allows the user to search for mathematical constructs (theorems) implicitly defined by the concepts of conditions and hypotheses found in mathematical propositions. A good example of a math search system is the MathWebSearch engine, presented before, limited only to searching mathematical subterms. A system which also employs automated reasoning tools on the query is the VSDITLU system (Verifiable Symbolic Definite Integral Table LookUp), which allows users to search for a definite integral with parameters and side conditions [AGLM99]. VSDITLU matches the user query against an already existing table of integrals and uses the PVS theorem prover to interpret the side conditions and unify them with the conditions listed for integral solutions in the table. Thus, the system is able to provide a custom result to the user, where all the constraints are met. The interesting part of this research is the parsing of user specified conditions and the matching and resolution against the existing conditions listed as possible branches of a solution, in order to deduce the correct answer.

2.2.6 Whelp

Apart from MathWebSearch, another content-oriented search engine for mathematical formulas is *Whelp* [AGC+04]. Developed at the University of Bologna as part of the MoWGLI project, Whelp provides four search functions for mathematical terms extracted from the mathematical knowledge database of the *Coq* [coq09] proof assistant. Users must introduce query terms of the Calculus of (co-)Inductive Constructions (CIC), with the possibility of wild cards, just like in MathWebSearch. Whelp uses an advanced type-based disambiguation algorithm to disambiguate the user queries and even offers users the possibility to select the right interpretation of an ambiguous query. The HINT search function of Whelp offers backward reasoning search for theorem proofs. Whelp only indexes the *constants* in its input and not the *variables*. The index of mathematical notions is created by extracting simple metadata describing hypotheses, conclusions and proofs from the sources and indexing that. Thus, by creating these extractors for different formats of mathematics which can be fully parsed, more corpora can be indexed.

2.3 Discussion

The research projects that I have presented in this chapter all make some contribution to the topic of natural language and/or mathematics processing. The closest ones to my proposed Applicable Theorem Search system are the math search engines (MathWebSearch and Whelp) and the complex natural language and chemistry formula parsing system (SciBorg). The SciBorg project performs information extraction by first transforming its databases into SciXML, a standard hierarchical structure XML format for scientific documents [CCMr$^+$06]. Similarly, the proposed Applicable Theorem Search system is an application which extracts natural language patterns from scientific documents in XML format. Such documents are found in the online Connexions repository and in the arXMLiv database. By employing pattern matching methods to extract the most frequent mathematical theorem formulations from the input texts, the Applicable Theorem Search system takes an approach which is located in the middle between the complete formal translation of the math discourse to *ForTheL* [VLPA08] and C. Zinn's approach of parsing the entire informal linguistic discourse [Zin04]. Then, by taking advantage of the generalization search function of MathWebSearch, it will retrieve theorems which fit the math term query to the user. The Whelp system comes very close to providing a theorem search utility, since it indexes a database of theorem proofs. Its limitations, however, lie in its restricted input syntax (which, for example, can not search for integrals) and its inability to do generalization search (as mentioned before, it does not store variables in the index). The Applicable Theorem Search system aims to be a search engine specifically designed for theorems and theorem-like formulations. Apart from offering generalization search, the system will also lead the user to the entire document where the match was found, in order to see the entire context of the theorem, function which Whelp can not perform due to the origin of its input.

Chapter 3

Theoretical Background

This chapter will provide the theoretical background for the main concepts used in this thesis and for the practical applications described in Chapters 4 - 6. In the following sections, I will first present a theoretical analysis of idioms in scientific texts and the search for idioms by pattern matching. Then, looking at the mathematical content of idioms, I will focus on the indexing of mathematic formulas and the search for mathematical terms. The final section of this chapter will bring the theories of idioms and mathematical terms together in a discussion about modelling natural language input by theorem patterns.

All the examples of idioms and mathematical formulas given in this chapter have been conceived for illustrative purposes and do not belong to any known corpus.

3.1 Natural Language Idioms in Scientific Texts

We start off with the assumption that authors of scientific documents employ similar natural language constructs in their rhetorics. We assume that formal statements (definitions, theorems, proofs, etc.) follow standard classical exposition patterns in natural language, even in unrelated scientific fields (like mathematics and biology, for example). For example, a definition may be found in a sentence of the pattern: *"We define X as Y"* or *"X is defined as Y"*, and a theorem may be expressed using a pattern of the type : *"Given X and Y we conclude that Z"* or *"If X then Y"*.

We call such natural language formulations which follow certain fixed word and syntax patterns: **idioms**. Idiom patterns give a natural language template for structured knowledge. **Idioms** are formed from fixed words or **keywords**, like *"define"* or *"if"* and **placeholders** like X or Y arranged in a given pattern. A language idiom actually expresses a semantic relation between the placeholders. For example *"We define X as Y"* translates to X relates to Y by the equality relation, or `eq(X,Y)`. Therefore, by identifying idioms, meaning is extracted from the text. The basic theory of idioms was first introduced in [AP09].

Considering common practice in English scientific articles and books and Claus Zinn's analysis of linguistic mathematical discourse [Zin04], we make the assumption that many scientific texts use the same types of idioms to express statements about mathematical constructs (therefore implicitly containing math formulas). This is probably because scientists are used to reading theorems (or definitions, etc.) in a certain format, and they will most likely also write them in the same format, as this helps them digest the information faster. This is not to say that they will follow a pattern specifically, word-by-word, but that they will generally keep a certain phrase structure and use the same keywords. Even though the English language is very rich in vocabulary and flexible in terms of formulations, the author makes the claim that a finite set of idiom patterns will match the majority of natural language formulations that bring about a particular class of desired knowledge.

In this thesis project, the idioms of interest are those containing mathematical formulas, so those patterns which bring knowledge about scientific fields using mathematics. Some illustrative examples of such idioms are presented below:

a) **Let** us assume $x, y \in \mathbb{N}$, **then** $x + y \in \mathbb{N}$.

b) We prove that **if** f is continuous on $[a, b]$, **then** f is bounded on $[a, b]$ **and** f is derivable.

c) Thus, we **conclude that** x is prime.

d) $C^k(\mathbb{R}, \mathbb{R}) \subseteq C^l(\mathbb{R}, \mathbb{R})$ **for all** $l < k \in \mathbb{N}$.

e) **Let** $f : A \rightarrow B$ and $g : B \rightarrow C$, **then** $f \circ g(x) = f(g(x))$.

f) **Let** the power spectrum $G(f)$ **be** defined as $G(f) = \int_{-\infty}^{f} S(f')$

As can be seen from the examples above and as mentioned before, the idioms are identified by the *keywords* they contain (written in boldface above and throughout the rest of this report), and the specific patterns in which they appear. A good heuristic definition of patterns based on keyword combination and order is the first approach taken in this project for idiom spotting.

3.2 Idiom Spotting

The task of finding idioms in the input documents is a classical Information Extraction problem applied to our corpus of XML documents with mathematical content. We can take an NLP approach to this task, as in the SciBorg project [Sci08], employing NLP tools like part-of-speech taggers, parsers, chunkers, etc. However, the main problem of employing these tools for scientific texts is the existence of math formulas, mixed in with natural language. The parsers employed by the SciBorg project mentioned in Section 2.2.4 are not natively designed for analyzing mathematical formulas and would break or give incorrect parse trees for a text with such entities inside. Thus, special care needs to be taken when attempting traditional approaches in Information Retrieval to process our input documents. For this reason, new custom methods have been employed for the task of extracting the natural language phrases with mathematical content out of the XML documents. The next two subsections discuss important considerations that need to be taken into account before attempting the spotting task.

3.2.1 Idiom Scope

We define **scope** of an idiom as the minimal syntactic context (made up of words, noun-phrases, sentences, paragraphs, etc.) needed to contain its full meaning. The idiom patterns usually extend to the syntactic level of a sentence or paragraph. This is natural, since sentences are stand-alone units of meaning in natural language. Hence, the meaning of an idiom will often-times be contained by the scope of a single sentence. As an illustrative example, the following idiom is self-contained in a sentence:

> **If** x is a root of the function f, **then** $f(x) = 0$ and $f'(x)$ is a local minimum.

However, the scope of an idiom can extend over more than one sentence, needing all the units to extract the whole idiom information, like in the following example:

> **Assume** $x \in \mathbb{C} \setminus \mathbb{R}$. We **define** x to be an *imaginary root of unity* **if** $\exists n \in \mathbb{N}$, s.t. $x^n = 1$.

Also, because of the length of mathematical formulas, the mathematical terms will most oftenly be found on a separate line. In terms of XML (which is the input format of the analyzed documents), this means a separation caused by a `<br>` tag, a `<div>` tag around the mathematical term or even (most oftenly encountered) the end of a paragraph `</p>`. Whatever the separating block, it causes the idiom scope to stretch across several XML nodes. For example, the idiom from the running example presented in Figure 1.2 stretches across several paragraphs:

> **If** given a recurrence relation of the form
>
> $$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$
>
> with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, **then**
>
> $$T(n) = \Theta\left(n^{\log_b a}\right)$$

In order to extract all of the information from the idiom, we must identify its entire scope, otherwise we might have incomplete semantics. Thus, in order to extract and idiom, we must first find its scope.

A good heuristic analysis should cover all of the scope cases listed above and any combination thereof. However, as in most research problems, we will start the analysis from the easiest, sentence case and extrapolate from there. .

An analysis of the idioms found at the **sentence level** needs to first isolate all the *interesting sentences* in the considered document. In the context of the current project, an *interesting sentence* (or paragraph, extrapolating) contains:

- at least a non-trivial mathematical formula
- at least an idiom *keyword*

All the interesting sentences extracted have math and a keyword which is linked to an idiom pattern. But a pattern usually contains more than one keyword. For example, the *If X then Y* pattern contains two keywords, *if* and *then.* In order to identify if a sentence contains an idiom, then all of the keywords must match in the right order (in other words, the interesting sentence must match the idiom pattern w.r.t. keywords and their order).

The **keywords** in an idiom define its meaning. For idioms containing multiple keywords, only one of them is considered the main keyword, while the others are secondary keywords. In practice, for any keyword spotting algorithm to run smoothly, the word that carries the most meaning should be chosen as the main keyword. The reason behind this is that an easy classification and identification of idioms is done based on this main keyword. And the more meaningful the word is, the less likely it is that a sentence which contains it will not be an idiom. Since various idiom patterns can be created around the same main keyword, we can classify idioms according to their main keyword. However, a more meaningful classification divides idioms according to the sematic relation that they express. The following section dwelves more into idiom classification.

3.2.2 Idiom Classification

The idiom **placeholders** are themselves of different **type**. The information that they capture makes up the composing parts of the semantic relation and thus, they have different roles depending on the idiom that they define. By defining idiom patterns according to the semantic relation they entail, we can create categories of knowledge which are to be extracted from the text. Thus, we can differentiate **idiom categories** (or classes) and their corresponding **types** of placeholders:

- **definition idiom**: *"Let X be Y"*, or *"Define X to be Y"*, where X and Y are the *definiendum* and the *definiens* (the order can be reversed)
- **theorem idiom**: *"If H then C"*, *"Let H then C"* or *"Given H1 and H2 then C1"*, where the H placeholders are of type *hypothesis* and the C placeholders of type *conclusion*
- **equivalence idiom**: *"X if and only if Y"*, where X and Y are of type *equivalent terms*
- **conclusion idiom**: *"Conclude X"*, where X is of type *conclusion.*

The same keyword-based approach for idiom spotting can be taken for all of the idiom categories listed above. Storing and retrieving them later (through user queries) however, is different, depending on the relation they define, the *number* of *placeholders* that compose a pattern and the number of *types* of placeholders found in an idiom. For example, the category of theorem idioms has only two placeholder types: **hypothesis** and **conclusion**, which can occur in a pattern with three, four or more placeholders: *"If H1 and H2 then C1 and C2"*. Thus, a *conclusion idiom* with only one placeholder type will be stored differently in a database than a *definition idiom* with two placeholder types.

All the idioms from the illustrative examples given above have the scope restricted to just one sentence pattern. We can call these **basic idioms**, since they are the basic building blocks for more complicated patterns. Putting two (or more) of these patterns together, we capture information with a larger scope and deeper semantics. For example, the construction

Let D be X. Then, if D' then C.

is a combination of a definition idiom pattern and a theorem idiom pattern, falling under the category of theorem idioms. The connection between the two patterns is given by the *D* placeholder, which appears again inside *D'*, thus bringing the two patterns together. We call such patterns **combined idioms**. Extracting semantics from such combined idioms recurses down to extracting the semantics of the basic idioms that compose them and then merging this information into one big scope. The merging step is non-trivial and thus makes extracting combined idioms a step in the direction of informal discourse understanding.

Sometimes, sentences which have an underlying idiom structure contain words which are not keywords, but are irrelevant to the meaning of the idiom. Examples of such words are: *"it follows"*,*"we can see"*, *"clearly"*, *"as expected"*, etc. We call these words **idiom - irrelevant**. In order to match sentences with idiom - irrelevant words, these words are either caught in the placeholders or are defined as keywords into new patterns. Neither of the options is desired, since the first introduces noise into the idiom components and the second requires the creation of many different very specific patterns. A special type of placeholder can be defined in order to capture the idiom - irrelevant parts of a sentence. We call it a **filler** placeholder, and it stands for content that is of no importance for understanding the idiom, and which will not be considered as part of its components. For example, the expressions "*We can conclude that C*" or "*Thus, we conclude that C*" both contain a beginning part which is idiom - irrelevant text. In order to cover them both with one idiom pattern, we would use

F conclude that C

where *F* stands for filler, or idiom-irrelevant content. An alternative to this approach is to use **regular expression idioms**, which provide even more generality than filler placeholders. Apart from hiding idiom-irrelevant words, regular expression idioms can offer *regular expression semantics.* For example, consider the regular expression:

$$\textit{If } (H_i \textit{ and})^* \textit{ H0 then C0 } (\textit{and } Y_j)^*$$

It is a theorem idiom with a variable number of placeholders and a non-trivial semantics in between the H_i hypotheses and the Y_j conclusions, respectively. Although this sort of idioms can get a lot of matches on text, the more complicated patterns can pose significant problems for the extraction and interpretation of the meaningful information from the patterns.

The practical application offering idiom retrieval and search functionalities entitled Applicable Theorem Search (described in detail in Chapter 6) only implements support for **basic theorem idioms***, as a proof of concept. The Idiom Spotter (see Chapter 4) has idiom extraction capabilities for basic idioms in all idiom categories with up to two placeholders types, and can easily be extended to handle idioms with more types. Thus, for the rest of the theoretical analysis, we will mainly focus on* **basic theorem idioms***, without loss of generality.*

Knowing the type of idiom we are looking for and the keywords that define it is all the information needed to start the information retrieval process. The next step is extracting the information out of the sentences which match the predefined patterns. This NLP task is not easy, due to the richness of natural language discourse and the difficulties involved in extracting only the relevant text + mathematics from the sentence scope. The next section describes some of these difficulties and analyzes a few idiom spottting methods which employ the general keyword and pattern matching approach.

3.3 Knowledge Extraction by Keyword and Pattern Matching

This section will describe the practical approach to idiom spotting based on keyword and pattern matching. First, I will outline some general difficulties encountered by this method and then present the Heuristic Pattern Matching variant of it, employed by the Applicable Theorem Search system. I will then also describe two possible alternative methods and their potential advantages.

3.3.1 General Difficulties

Spotting the keywords from a sentence in a specific order to establish that they correspond to an idiom pattern is not a difficult task. However, extracting only the relevant information out of the sentence or out of the "chunk" of words in between the keyword delimiters can be considered a challenge. For example, take the formulation

> **If** we consider $|x| > n$ and we have shown before that $n \geq 0$, **then** normally we have that $x \neq 0$.

By just looking at the keyword delimiters and extracting everything in between, we find the hypothesis to be *"we consider $|x| > n$ and we have shown before that $n \geq 0$"* and the conclusion to be *"normally we have that $x \neq 0$."*, instead of just the relevant $|x| > n$, $n \geq 0$ and $x \neq 0$. The **idiom - irrelevant words** caught in these parts are inherent to natural language and need to be carefully sifted, in order not to lose semantic information from the idiom.

Another problem that comes up is the **separation of math blocks** in natural language sentences. When authors write scientific papers, they tend to use natural language formulations which sometimes hinder the cohesion of the mathematical formulas inside. For example, in the sentence

If $n \in \mathbb{N}$, **then** $\frac{1}{n} \to 0$ and, we can now prove our claim that $f(n) < 0$.

The conclusion is actually a pair, formed from $\frac{1}{n} \to 0$ and $f(n) < 0$. However, these two formula blocks are separated by the words *"and, we can now prove our claim that"*. Extracting both conclusions separately from this sentence with a simple idiom can prove to be a difficult task for a pattern matching approach, since very general and comprehensive patterns are needed. Also, the natural language formulations which tie different parts of a conclusion (or hypothesis) together can vary very much and easily fall outside of a pattern. Regular expression idiom patterns might help, but only for extracting all the math blocks and discarding the text, losing all possible semantics given by the natural language which connects them.

One of the most complex problems related to idiom spotting involves the **scope** of mathematical expressions. This problems appears in the case of combined idioms, where even though it looks like the whole idiom itself might be self-contained in a sentence, the formulas appearing in it might be defined before (or after), thus enlarging the scope. For example:

Let A **be** a diagonal 2 X 2 matrix $\{a_{00}, a_{11}\}$. Then, we **define** $P_A(x) = (x - a_{00})(x - a_{11})$ to **be** the characteristic polynomial of A.

In the *"Define X to be Y"* idiom above, the definition of P_A is incomplete without the knowledge from the first sentence. Being able to gather the information from the first idiom and then combine it with the second idiom to extract joint semantic information needs a *scope* large enough to contain both and *context - based disambiguation* techniques, to unify all occurences of A. Thus, the simple keyword and pattern matching technique is not enough to extract semantics from combined idioms.

We will now take a look at a simple heuristic pattern matching method for idiom spotting, based on keywords and their order in a sentence. We will then also analyze potential gains by a syntactical analysis approach an an aproach using Discourse Representation Theory [Kam95] to extract more semantics from the input text.

3.3.2 Heuristic Pattern Matching

The Heuristic Pattern Matching approach for idiom spotting is a simple, and yet comprehensive attempt to extract idioms from text. A heuristic database of patterns is created and used as the basis for pattern matching by keyword search. First, a run through the entire text of an article is performed. Then, the sentences which contain keywords are stored and matched against the predefined patterns. If they correspond to a particular pattern, they are analyzed and the relevant idiom parts (for e.g. conclusions and hypotheses) are extracted, as shown in more detail in [AP09].

The exact meaning of *pattern matching* in this algorithm, is that the sentence is analyzed word by word and the ordered list of idiom keywords must be a subset of the ordered list of sentence words in order for the sentence to match a certain idiom pattern. The main idea is to extract all the text (and math) that is found in between the keywords in the pattern (and will by definition match a placeholder). For example, let's analyze the extracted information from the idioms below:

1. **If** we prove that $x^2 < 0$, **then** we know for sure that $x \notin \mathbb{R}$.
2. **Let** us assume that $f(x) = xe^{ax}$; **then** the corresponding integral is

$$\int f(x)dx = \left(\frac{x}{a} - \frac{1}{a^2}\right) e^{ax} + C.$$

3. **If** z is any non-zero complex number for which the series $\sum_{k=0}^{\infty} a_k z^k$ converges, **then** it follows that $\lim_{t \to 1^-} G_a(tz) = \sum_{k=0}^{\infty} a_k z^k$.

In example 1. (an *If X then Y* idiom) the extracted hypothesis is *"we prove that $x^2 < 0$"*, while the conclusion is *"we know for sure that $x \notin \mathbb{R}$"*. In example 2. (*Let X; then Y*), the hypothesis string is *"us assume that $f(x) = xe^{ax}$"* and the conclusion is *"the corresponding integral is $\int f(x)dx = \left(\frac{x}{a} - \frac{1}{a^2}\right) e^{ax} + C$"*. As you can see, a lot of words captured in these hypotheses and conclusions are idiom - irrelevant and should be left out (e.g. *we prove*, *we know for sure*, *it follows*). However, looking at example 3., the words in between the math blocks in the hypothesis *"z is any non-zero complex number for which the series $\sum_{k=0}^{\infty} a_k z^k$ converges"* are indispensable for the complete meaning of the idiom. These words can not be left out without breaking the semantics of the hypothesis, as opposed to the first two examples in Section 3.3.1. Making the distinction between these two cases is the difficult task, which must be solved by the idiom patterns provided. Thus, the Heuristic Pattern Matching approach creates the need for well-specified patterns, that match as many of the natural language formulations as possible and catch as few idiom - irrelevant words as possible.

We represent the idiom patterns, which are matched against the scientific documents, in simple text format, where placeholders are encoded by capital letters followed by a number. For *theorem idioms*, the following notation is used: for **hypothesis** - the letter **H**, for

conclusion - C. We also number the identifiers, since we can have different hypotheses and different conclusions in the same idiom. Just consider these two cases:

1. **Let** $x \in \mathbb{R}$ **and** $y \in \mathbb{Q}$, **then** $xy \in \mathbb{R}$.
2. **If** $x \in \mathbb{N}$ **and** $y \in \mathbb{R}$, **then** $f_{select}(x) = 0$ **and** $f_{select}(y) = 1$, respectively.

In both cases there are multiple hypotheses. The same holds for conclusion placeholders; we could have several of each, depending on the idiom. The interesting question in such sentences is to identify which conclusion is attached to which hypothesis, if that can be distinguished. In the first example, the same conclusion is attached to both hypotheses while the second case is slightly more complex. We use numbers to solve this problem. Hypotheses and conclusions which have the same numbers are attached on to each other. The first example above would match the idiom *"Let H1 and H2 be C1"*, where, because there is no *C2*, both hypothesis are attached to the only conclusion *C1*. The second example would match the pattern *"Let H1 and H2 then C1 and C2 respectively"*. Note that one could attach several conclusions to one hypothesis or many hypotheses to a conclusion or even many hypotheses to many conclusions. The latter is the most general case, when a distinct connection between a particular hypothesis and a particular conclusion can not be determined. Theoretically, this is represented by considering that the idiom relation holds for any pair **(hypothesis, conclusion)** taken from the cross product of the sets of **hypotheses** and **conclusions**.

The question of determining a direct connection between a hypothesis and a conclusion is a fine granularity of the *theorem idiom* problem. Most of the time, it requires complete formula independence between different conclusions (the mathematical variables appearing in one conclusion must not appear in another conclusion) and the existence of meaningful keywords like *respectively* in the sentence.

Another issue that we encounter with this slightly more general form of idiom representation is the decision of when to use one idiom pattern over another. For example, a sentence that would fit the idiom *"Let X1 and X2 be Y1 and Y2 respectively"* can also in principle fit *"Let X1 and X2 be Y1"* or *"Let X1 be Y1"*. There is an obvious need for the ranking of similar idioms according to their generality. In the implementation presented in [AP09], a simple solution is used, where idioms are ranked according to their length. The shorter the idiom is, the more general it becomes. If it happens that several idioms fit a sentence, the least general one is chosen (the longer one).

The Idiom Spotter system, presented in Chapter 4, provides the first implementation of the Heuristic Pattern Matching approach for idiom extraction described in this section. It processes XML documents and uses easily (re)definable idiom patterns in the format described in this section to retreive semantic information. While this is still a semantically shallow approach to knowledge extraction, the Idiom Spotter provides one of the first attempts to extract mathematics and text from XML documents in a semantic manner. For a detailed analysis and experimental results, please see Chapter 4.

For the sake of completeness, let us see what the Heuristic Pattern Matching method outputs in the case of the running example from Figure 1.2. It is easy to identify the *conclusion* and the *hypothesis* parts of an *"If H then C"* theorem idiom:

- **Hypothesis**: a recurrence relation of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$
- **Conclusion**: $T(n) = \Theta\left(n^{\log_b a}\right)$

Challenges and Improvements

The main challenge of this approach comes from defining the idiom patterns in a smart way (such that only the useful information is being captured and as few idiom - irrelevant words as possible get caught in the components) and from extracting the semantic information from the placeholders. Knowledge about the most usual patterns encountered on these scientific texts and their structure can be gained by experimentation on the available corpora. Also, some more insight can be gained by symbolically attributing a fixed part of speech to a mathematics block and then running a part of speech tagger (POST) over the sentence. Consequently, only that one predefined part of speech would be extracted from the text and replaced with the initial math formula.

A way to tackle some of the challenges mentioned in Section 3.3.1, inherent to this approach, could be to interpret the entire sentence and extract its syntactical (or logical) structure before trying to identify the hypotheses and conclusions inside. A syntactical approach could be to interpret a sentence into a parse tree and then analyze its syntactical components before trying to extract the idiom. A logical approach, using Discourse Representation Theory [Kam95], could process the sentence and turn it into Discourse Representation Structures first, which offer semantical relations between the parts of a sentence. These two methods could potentially bring better results for idiom spotting, and we will shortly analyze what their possible advantages in the following two sections.

3.3.3 Syntactical Structure Analysis

Another approach to idiom spotting by keyword and pattern matching is the syntactical analysis method. This method could provide a way to escape the problems created by idiom - irrelevant words by analyzing the sentence structure at a higher level. By using a syntax parser on reoccurring idiom stuctures, the syntactical roles of the keywords and placeholders in an idiom are defined into a pattern, and can be used as labels for hypotheses and conclusions in a theorem idiom. This method would still fall under the category of keyword and pattern matching approaches, as we would first perform main keyword search (and parse only the sentence scope of the results) and then apply syntactical patterns to the resulting parse tree in order to find an idiom match.

This method highly depends on the parsers used and the syntactical patterns sought after. For example, if we pass the sentence

If x is prime, **then** x is a natural number.

through a simple regular parser, we would get an analysis similar to: (NP x) (VP is (NP prime)) and (NP x) (VP is (NP a natural number))[1], where NP is a *noun phrase* and VP is a *verb phrase*, and the two parse trees represent the hypothesis part and conclusion part, respectively. It seems intuitive that a hypothesis or a conclusion would be a noun phrase in a sentence or a combination of a noun phrase and a predicate. Indeed, in our experiments with mathematical formulations, this is very often the case. However, the problem is that the idioms that we focus on contain mathematical formulas. Since most parsers are not trained on text with mathematical formulas, the parsing will fail and return erroneous results. Thus, in order to be able to run such parsers on scientific texts, all the non-trivial mathematical terms need to replaced with a generic natural language equivalent. For example,

If we take $n \geq 0$, **then** we know that $\sum_{i=0}^{n} i = \frac{n \cdot (n+1)}{2}$.

would be transformed into

If we take *math term A*, **then** we know that *math term B.*

Once these replacement words or sequence of words are in place, the sentence can be parsed and the resulting syntax tree analyzed. By knowing the part of the sentence that we are looking for, we can extract just the relevant information from the idiom, based solely on the syntactical role that the math replacement plays in the sentence structure. For the above example, both the hypothesis and the conclusion are NPs preceded by a VP (and a VP plus *that* for the conclusion). This can be transformed into a syntactic rule or pattern, specific for *If X then Y* idioms. An example of such a pattern (created with the help of the CMU Link parser [Lin09]) would be:

If (S (NP we) (VP take (NP [*hypothesis*]))),
then (S (NP we) (VP know (NP [that *conclusion*])))

The words *"we"*, *"take"* and *"know"* would not be part of the pattern but are shown above for understanding how this pattern fits the afore-mentioned *if H then C* idiom. This syntactical idiom pattern would make use of parse tree cuts in order to find matches in the input sentences. However, this is only one positive example. The discovery of a good generic math replacement that would work in most cases and would keep the syntactical role of the sentence is actually a difficult task and should be approached very carefully. In sentences where the mathematics is more complicated, we can find that replacing the formulas with just one noun or a noun with a determiner is not enough, because the predicate is missing. In the example below, we can easily see that a simple replacement with "math term A" is not enough:

[1]Output of the CMU Link Parser [Lin09]

If $x \in \mathbb{C}$, **then** $x^2 \in \mathbb{R}$.

If we transform it to *"If math term A, then math term B"*, then it becomes obvious that the sentence is lacking predicates and makes no syntactic sense. In this case, a simple parser would probably identify *"some"* as an NP and *"math"* as a VP, making the syntax tree unusable. Thus, for this particular situation, a generic replacement that contains a verb as well would fit better.

This method for idiom spotting could be more accurate than the one described in Section 3.3.2. By only extracting the hypotheses (or conclusions) which are NPs, for example, we could eliminate the false positives found by the heuristic method and restrict the syntactical scope of the idiom components, potentially sifting a lot of idiom - irrelevant words. A combination of the syntactical analysis method with the heuristic analysis method can also be foreseen, where the syntactical analysis matching is only done on local subtrees, thus performing a sort of syntactical typing of the idiom components.

The choice of the generic word replacements for math is correlated with the choice of the parser used for syntactic analysis. For the illustrative examples given in this chapter, I have used the CMU Link parser described in [ST93]. The Link Parser follows the Link Grammar, which creates different types of links in between pairs of words and was chosen for the simpler parse trees outputted, useful for exemplifying base concepts. However, at the time of this thesis, there are numerous research projects which focus on parsers, ranging from shallow to very deep levels of analysis. The choice of the correct parser to use has to be done very carefully, considering that a deep level of analysis might be unneccessarily complicated, while a shallow level might not be specific enough for idiom structures.

The syntactical analysis method of idiom spotting has not been implemented in the practical IDIOM SPOTTER *application yet, making all considerations given in this section purely theoretical. The implementation and testing of this method is left for future research on this topic.*

3.3.4 Discourse Representation Theory Analysis

A third approach to finding idioms could be to extract the logical structure of the scope before any pattern matching is performed. Similar to C. Zinn's attempt at understanding informal mathematical discourse [Zin04], we could use Discourse Representation Theory (DRT) [Kam95] for this extraction. A practical implementation of DRT is given by Johan Bos's Boxer tool [Cur07], which generates semantic representations of natural language text based on a Combinatory Categorial Grammar. It outputs the underlying logic behind sentences in the DRT formalism, and produces corresponding Discourse Representation Structures(DRS) as a final output.

By looking at the structure of the DRS output of Boxer on predefined idioms, we can again identify some patterns. This time, they will be DRS patterns and the relations involving the idiom components in such structures can be settled. These pattern rules can then be employed to spot idiom structures on input text.

We would start off the idiom spotting with keyword search to point to the sentences which contain idioms, and extract their scope. Then, the preprocessing step that needs to be taken for DRT analysis is to again replace the mathematical formulas by a proper generic word that would fit the semantics of the sentence in most cases. Only then, we can run Boxer over the result to create the DRSes that represent the whole logical structure of the sentence with the mathematics replaced out of it. We would then match the retrieved DRSes against the pattern rules previously identified, hoping to find the same structure. For this algorithm to work, we have to make the assumption that replacing the mathematics with generic words before the DRT analysis does not affect the logical meaning of the sentence.

For example, let's take a look at a natural language definition idiom below, where the definiendum is *"an affirmation"* and the definiens is *"a lie"*:

> **Let** an affirmation **be** a lie.

Upon analysis, Boxer [Box] returns a DRS with the following relations, among others: `affirmation(x1), let(x2), patient(x2,x1), lie(x4), eq(x1,x4)`[2]. A conclusion that could be drawn from here, for example, is that in a *Let D be C* idiom, the definiendum *D* is a `patient` of *let* and the definiens *C* will be in a `eq` (equality) relation with *D* (which makes logical sense). This can be stored as a pattern rule.

Then, a scientific text with the same idiom structure where the mathematical formulas have been replaced by words would be passed through the same system in the form:

> **Let** math1 **be** math2.

Boxer returns a very similar DRS as before, with `math1(x1), let(x2), patient(x2, x1), math2(x4), eq(x1, x4)`. By applying the rule determined above, we can quickly infer that *math1* is the definiendum and *math2* is the definiens. A positive argument for using this sort of approach for idiom spotting is that on the input *Let some math1 be some math2* or *Let some math1 be some other math2*, the above relations do not change. Thus, the Boxer analysis method can prove to be a very elegant way to get rid of idiom - irrelevant words.

Choosing suitable generic words is very important to this approach as well. The examples given in this section were just fragments of Boxer analyses at [Box09] on simple inputs, for the purpose of introducing the DRT idiom spotting approach.

The Discourse Representation Theory method of idiom spotting has also not been implemented in the practical IDIOM SPOTTER *application yet, making all considerations given in this section purely theoretical. The implementation and testing of this method is left for future research on this topic.*

[2]A part of the analysis obtained at the *Boxer* home page [Box09]

Even though the syntactical analysis and the DRT idiom spotting methods employ more semantics than the heuristic method, it is questionable, without empirical results, if either of the two is more efficient at finding idioms than the simple pattern matching approach employed by the Applicable Theorem Search system.

Continuing this chapter, we will switch from the natural language - oriented idiom analysis to the theory behind the mathematical terms found in idioms, specifically the semantic of these terms and the way they are indexed and searched for.

3.4 Mathematical Variables and Applicable Formulas

The idioms that we focus on in this thesis project contain mathematical terms, or formulas. The semantics of these terms can differ, depending on their mathematical and linguistic context. I will now present some general considerations about the type of terms we can encounter in mathematical formulae, the ones that we are interested in for the scope of this project and how to combine their semantics with that of idioms for extra added value.

The Applicable Theorem Search system is a tool which aims to bring applicable theorems to the user, on the basis of a mathematical search query. In order to understand how we can determine whether a certain theorem is applicable, we must first of all define the concepts of *bound variables* and *universals*.

In a mathematical formula, we can usually find *operators* (like $+$ or $\int$), *constants* (like 2 or e) and *variables* (like x or a). The mathematical variables found in a formula usually have a placeholder role, referring to one (or more) constants which can be later replaced in the formula without changing the meaning or truth value. For example, in the formula $x \in \mathbb{R}, x^2 = 1$, x can later be replaced with 1 or -1, and in $\forall x, x^0 = 1$, x can be replaced with any constant. The syntactical way to specify what we can replace a variable with in mathematical formulae is to use *quantifiers*, like $\forall$ and $\exists$. Quantifiers settle or *bind* variables with respect to what they stand for in a mathematical formula.

Given the scope of a mathematical formula with variables, a **bound variable** is any variable that has a quantifier or binding operator associated to it (quantifying it), specifying what the variable can be instantiated with. Similarly, in a given mathematical scope, a **free variable** is a variable which is not quantifed. For example, taking the illustrative mathematical formula out of its context:

$$\forall x, \exists y \ s.t. \ f(x) = y^n + a$$

x and y are bound, while f, n and a are all free variables.

Operators which bind variables are $\int$, $\sum$, etc. In order for a mathematical formula to make logical sense, all of its variables must be bound, otherwise there is ambiguity about what they can be instantiated with. This is usually done in the mathematical (or linguistic) context (scope) of the formula. Usually, when given a mathematical formula with free variables, expanding its scope finds the missing binders.

When harvesting mathematical terms with variables, the question of deciding whether they match as applicable theorems to a query term reduces to the question of whether the constants in the query term can be instantiated for the variables in the indexed math formulae. The answer to this question is given by the quantifiers which bind the variables in the index, specifying exactly the range of values that can be given to them. Storing the scope and all the preconditions of bound variables in the index and then computing whether the constants in the input query match those contraints would require typing of the math formulas (to decide on conditions like $\forall x \in Q$) and a computational engine to determine constraint satisfaction. This is a non-trivial computational approach that is not performed by the Applicable Theorem Search or the MathWebSearch systems. The reason for this is that the constraints on the indexed bound variables need to be extracted from informal mathematical discourse, which is not possible at the time of this thesis (see [Zin04]). The Applicable Theorem Search system's approach to indexing applicable theorems is to only retrieve *universals* and to abstract away from type.

Universal variables are those variables bound by an universal ($\forall$) quantifier. These variables can usually be instantiated by a wide range of constants (e.g. $\forall k > 0$, $\forall x \in \mathbb{R}$) and constitute the best type of applicable term for our purposes. By abstracting away from the type ($\mathbb{N}, \mathbb{Z}, \mathbb{R}$, etc.), we add the mathematical terms with these universal variables to the index and then instantiate them freely at search time, to any constant. This functionality is currently performed by the MathWebSearch system and provided as a *generalization search* for the user. Thus, an **applicable formula** is any formula with universals that can be instantiated to a user query. This approach needs to distinguish the existentially bound variables from the universally bound variables, since only the latter are added to the index.

The status of variables as existential or universal can be defined both in mathematical and linguistic context. For example, the following two declarations are semantically equivalent:

- $\forall x, \exists y.\ 4^x = 2^y$
- For all x, there exists y, s.t. $4^x = 2^y$.

The above statement may occur frequently as often in either form, but is processed differently by automated tools of knowledge retrieval. In the first example, x and y are bound within the scope of the mathematical formula, and it is clear that x is universal. In the second example, both x and y are free variables in the mathematical formula $4^x = 2^y$. The linguistic context binds them through words like "for all" and "there exists". Thus, the needed scope of the mathematical expression is the whole sentence, while the scope in the first example is only represented by the mathematical formula. Looking only at mathematical formulas, the MathWebSearch system has no way of knowing the status of x and y, without understanding the linguistic context. Thus, the compromise that has been made so far in indexing such formulas is to consider all free variables as universals

(MathWebSearch 0.4, presented in [KAJ+08]). Although this assumption creates false positives for variables which are existentially bound in the linguistic context, it allows for generalization search in a case where no context information from outside the math formulae can be extracted. The developers of MathWebSearch claim that this assumption is more often correct than false. Formulations which do not specifically bind all variables are encountered frequently in scientific texts for the sake of succintness. For example the Master theorem in Figure 1.2 leaves T and n free, even in the syntactical linguistics context. This is because the reader can imply their meaning from the meta-context. The Applicable Theorem Search system, though, has the right tools to bring the mathematics and natural language context information together.

Idioms can provide the right mechanism to extract information about the status of free variables from the linguistic context. Considering that the binding of variables in natural language is achieved by the usage of certain words, we can define **context idioms** as idioms which bring linguistic context information about the mathematical terms in their components. Examples of such idiom patterns would be:

- *For all U let U'*, where all the variables in U are universal.
- *Given H there exists E such that E'*, where all the variables in E are existentially bound and should not be considered as universal in *E'*

The first example is an applicable theorem, with universals declared by the language. However, the second example is more important, as it shows how idioms can use the linguistic context to decide that the construction is not an applicable theorem, while a restricted scope analysis at the formula level of *E'* would probably decide the opposite. Making no assumptions about free variables in the mathematical formulae, context idioms are a much simpler (and plausible) way to decide on universals than the complete understanding of the informal mathematical discourse.

With such context idioms, the Applicable Theorem Search system can make the connection between the concept of universal variables in mathematics and logics and the linguistically defined universals, thus extracting **combined semantics** from both natural language and mathematics processing. A foreseeable algorithm for deciding on applicable theorems is to extract the whole *context idiom* scope and first look at the mathematical formulas inside. Then, we would look at the information brought by the context idiom and try to find linguistic binders for all the variables which are not explicitly bound in the mathematical formulas. Then all the leftover free variables are subjected to a choice. We could assume that all the free variables are universals and add them to the applicable theorem index, which would create an index with false positives. The second option is to assume that all free variables are existentially bound, and not add them to the index, which would create an index of only well-defined (in terms of the mathematical rigour of binding all variables) applicable theorems. We can even foresee an option to create both indeces and leave it up to the user to decide which assumption to make.

Version 0.4 of the MathWebSearch *system provides generalization search based on indexing* *free variables as universals. Building on top of it, the* Applicable Theorem Search *system provides the same functionality in the search for applicable theorems. The* Idiom Spotter *system does not yet provide support for context idioms. The processing of such idioms first needs an extended context disambiguation of variables (see Section 5.2.7) and is a topic for future work on both* Idiom Spotter *and* Applicable Theorem Search *systems.*

Restricting our analysis to just mathematical terms, the next section will provide a description of a mathematical-term indexing algorithm.

3.5 Mathematical Term Indexing

Having presented the theoretical considerations about the semantics of mathematical terms in the previous section, we will now analyze their path from the input documents to the user. The first stage is indexing, the process of extraction from the XML documents and storage in a database index.

In order to provide a service that offers mathematical subterm querying and searching capabilities, the representation in which the mathematical formulas are stored in a database index must be carefully chosen. The MathWebSearch system offers a multitude of mathematical querying capabilities, (*instantiation*, *variation*, *generalization* and *unification* search), all based on the same representation of mathematics in a *substitution tree.*

Substitution-tree Indexing

A method of aggregating mathematical formulas in an index based by their structure similarity is to add them to a search tree. This way, the tree holds the general structure of a mathematical formula and searches can be easily and quickly made in the tree following this structure. The method of *substitution-tree indexing*, described in [Gra96] is an efficient subterm indexing technique, employed by the MathWebSearch system. It consists of a single tree for the whole index, holding a general formula structure into which all possible terms fit. As the name already suggests, the tree has substitutions in each node and satisfies the following conditions:

- Each node is either leaf or has at least two children.
- The substitution at the root of the tree is of form $\{\mathcal{Q} \rightarrow \tau\}$ where $\mathcal{Q}$ is some distinguished element of alphabet and τ is some term.
- Let σ_1, σ_2, ... σ_k represent substitutions along the path from root to a leaf, σ_1 being the substitution at the root of the tree. Then
  - successive application of substitutions from root to a leaf results in one of the indexed terms i.e. $\sigma_1 \circ \sigma_2 \circ ... \circ \sigma_k = \{\mathcal{Q} \rightarrow t\}$ where t is one of the indexed

terms.

– a substitution σ_i substitutes different elements i.e. $DOM(\sigma_i) \neq DOM(\sigma_j)$ for $i \neq j$.

Figure 3.1 shows a typical index for the terms $h(f(z,a,z))$, x, $g(f(z,y,a))$, $g(f(7,z,a))$, and $g(f(7,z,f))$. For clarity we also present the term we get if we apply all the substitutions starting from the root of the tree. The subterms `@integer` are the generic subterms, which are to be instantiated by a substitution. A leaf has no more generic `@` terms, as no more substitutions can be applied to it. For further details and the term insertion algorithm, see [KS06].

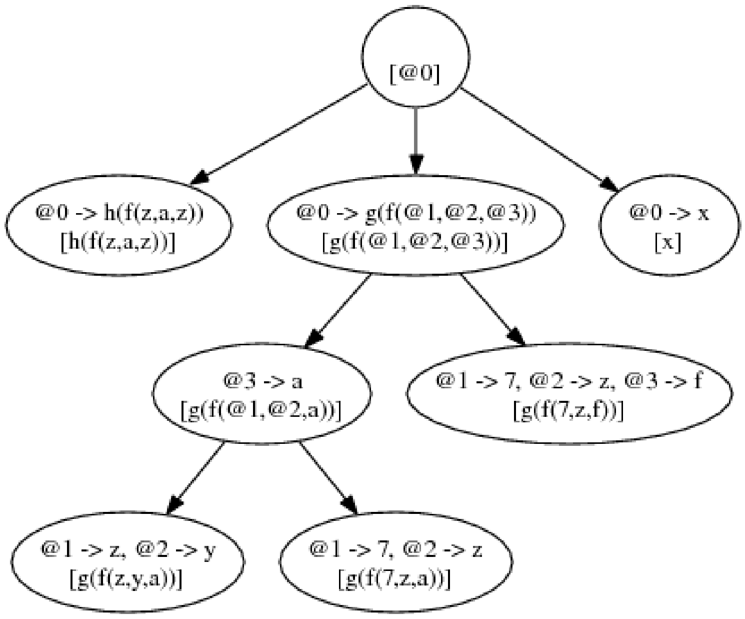


Figure 3.1: An Substitution-tree Index with Five Terms [KS06]

As used here, we will use the `@` notation in the following sections of this thesis to denote **generic terms**. At any level in the parse tree of a mathematical formula, a *generic* term `@0` is a placeholder or wild card that can be instantiated by any other term, through a substitution. Generic terms are similar to mathematical variables, defined in Section 3.4, with the exception that they can be instantiated by anything and have no type. A generic term is usually represented by an `@` symbol in front of the term name. For example, `@x` shows that x is the name of a generic term. The substitution-tree index in Figure 3.1 uses the generic term notation as a technical device to show terms which can be further substituted.

3.6 Mathematical Term Search

Using the substitution-tree as an index, different type of searches can be made on its structure, taking advantage of the locality of formulas with similar structure in the tree. I will now present different types of searches which are offered by MathWebSearch 0.4 and show how they work on terms stored in the *substitution-tree* index, as this is important for understanding the functionality of the Applicable Theorem Search system. The implementation of these methods was made possible by Ioan Şucan, Constantin Jucovschi and myself and presented in [KAJ+08]. The mathematical examples given in this chapter try to follow the running example from Section 1.2 and were created for illustrative purposes.

3.6.1 Classical Search

A **classical search** for a mathematical term is a search where the query formula is fully defined w.r.t to structure and term names, leaving no generic terms to be substituted. A mathematical definition for it would be:

Given an index $\mathcal{I}$, let τ be a search query term. Then, a **classical search** for τ returns:

$$C(\tau, \mathcal{I}) := \{t_i \in \mathcal{I} \mid \tau = t_i\}$$

For example, the query for:

$$T\left(\frac{n}{b}\right)$$

would possibly retrieve

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

from the index, as the subterm match is complete. However, it would not retrieve a term like

$$H(x) = aH\left(\frac{x}{b}\right) + f(x)$$

which is semantically equivalent, given that n is a bound variable which can be renamed in the index.

Classical search is not interesting from the semantic point of view, but it comes at no extra cost and it is automatically included in other, more complex search types.

3.6.2 Instantiation Search

Instantiation search is the search method that uses the concepts of *generic* terms and *substitutions*, employed in creating the structured index, to retrieve fully defined mathematical terms. Instantiation search takes queries containing wild cards (or generic variables) and compares them to the index. The index then searches for a node that the query subterm matches and then returns all of its children, with the substitutions that were made, from the matched node to each of the leaves. The mathematical definition is:

Given an index $\mathcal{I}$, let τ be a search query term. Then, an **instantiation search** for τ returns:

$$I(\tau,\mathcal{I}) := \{t_i \in \mathcal{I} \mid \exists\sigma \text{ s.t. } \sigma(\tau) = t_i\}$$

For example, assume the user enters a search query of

$$@f\left(\frac{@a}{@b}\right)$$

Then, possible results are:

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with the substitutions $@f \to T$, $@a \to n$, $@b \to b$
- $g(x) = \sin(\frac{\log x}{2}) + \pi$, with the substitutions $@f \to \sin$, $@a \to \log x$, $@b \to 2$

As you can see, the instantiation of generic terms is done with indexed terms of any depth, as in the second example, where $@a \to \log x$.

If the user were to enter the query

$$@f\left(\frac{@a}{@b}\right) + \pi$$

then only the first result would have been returned. This shows that classical search is comprised in instantiation search. Instantiation search is useful when looking for the general structure of a formula but not remembering details about coefficients or names of terms. Instantiation search was the first search functionality offered by MathWebSearch 0.1 [KS06] and it was solved by keeping constants in the index and using generic terms to play the role of universals in the query.

3.6.3 Variation Search

Variation search implements the concept of *α-renaming* (or *α-conversion*) from *λ-Calculus*. Usually, *α-renaming* allows bound variable names to be changed, but keeps the semantics equivalent. For example, an alpha-conversion of $\lambda x.x$ would be $\lambda y.y$, keeping the expression equivalent. MathWebSearch variation queries select all the bound terms from the index which are equal to the search term after possible renamings of variable names.

Given an index $\mathcal{I}$, let τ be a search query term. Then, a **variation search** for τ returns:

$$V(\tau,\mathcal{I}) := \{t_i \in \mathcal{I} \mid \exists\sigma_\alpha \in \Sigma_\alpha \text{ s.t. } \mathbf{intro}(\sigma_\alpha) \cap \mathbf{free}(t_i) = \varnothing \wedge \sigma_\alpha(\tau) = t_i\}$$

For example, the query for:

$$e^a \sin(a+b)$$

would possibly retrieve

$$\int\int e^x \sin(x+y)dxdy$$

with the substitutions in the index $@x \to a$, $@y \to b$.

Relating to generic terms, a variation search is a substitution which considers all variables in the query as generic terms and α-substitutes them with bound variables from the index. These bound variables in the index are usually terms of depth 1. Thus, it is a search for non-universals from the index. The variation search option can be useful when the user knows the full mathematical structure of the searched-after term, but does not want to deal with the large variety of names for bound variables in mathematical texts, which tend to change from author to author.

3.6.4 Generalization Search

Generalization search works in the opposite direction to Instantiation Search. Basically, the user enters a query with some concrete values, and the search engine tries to apply substitutions to terms with **universal variables** in the index, in order to get the query term. It is called *generalization search* because, practically, the concrete values in the query term are instantiated into a more general formula in the index. Formally,

Given an index $\mathcal{I}$, let τ be a search query term. Then, a **generalization search** for τ returns:

$$G(\tau, \mathcal{I}) := \{t_i \in \mathcal{I} \mid \exists\sigma \text{ s.t. } \sigma(t_i) = \tau\}$$

As an example, we have the running example use case, when a scientist looks for a practical mathematical term:

$$C(1) = 9C\left(\frac{1}{3}\right) + r(1)$$

and the search engine returns the general mathematical formula

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

which is stored in the index with universals (denoted here by @) as $@T(@n) = @a@T\left(\frac{@n}{@b}\right) + @f(@n)$ and matched with the substitutions $@T \to C$, $@n \to 1$, $@a \to 9$, $@b \to 3$, $@f \to r$. Note that even though we used the same @ notation for them, generic terms and universal variables are two different concepts in the index. Universals are mathematical concepts stored at the leaves of the substitution-tree, while generic terms are technical constructs stored in intermediate non-leaf nodes in the tree showing that the terms at the particular tree level can be further substituted. The reason why we use the same notation for both is to show that both are instantiated by substitutions at search time.

Generalization search is the most useful type of search for the Applicable Theorem Search system, as it can find all general formulas or theorems which can be applied to the searched term. It provides a way to extrapolate from a use case to a general theory.

3.6.5 Unification Search

By definition, the **unification** of two terms is the common substitution which, when applied to both terms, makes them equal. For search, the two terms are the query term and the indexed term that matches. **Unification search** is the search type that tries to instantiate generic variables in the query and universal variables in the indexed terms in order to find a match. This is the most general type of query, which contains all the other types mentioned so far. Formally,

Given an index $\mathcal{I}$, let τ be a search query term. Then, a **unification search** for τ returns:

$$U(\tau, \mathcal{I}) := \{t_i \in \mathcal{I} \mid \exists \sigma \text{ s.t. } \sigma(t_i) = \sigma(\tau)\}$$

Consider the example when the user searches for the term with generic variables

$$@1 \int_0^\pi (f+g)^{@2}(t)dt$$

and the search engine finds in the index the term

$$\frac{1}{@3} \int_0^{@3} (@4)^2(@5)d@5 = \sum_{k=-\infty}^{\infty} \|@c_k\|^2$$

They are unified by the substitutions $@1 \to \frac{1}{\pi}$, $@2 \to 2$, $@3 \to \pi$, $@4 \to (f+g)$, $@5 \to t$ into the term

$$\frac{1}{\pi} \int_0^{\pi} (f+g)^2(t)dt = \sum_{k=-\infty}^{\infty} \|@c_k\|^2$$

This type of query is useful when the user has a term which needs instantiation in some parts and generalization in other parts. Since all of the searches mentioned so far can be expressed in terms of unification search, we can combine them into complex queries.

Along with the introduction of unification search in MathWebSearch 0.4 [KAJ+08], variation and generalization were also added as search services. This changed the representation of terms in the index, where now universals are stored and are retrieved by searching for constants in the query.

By presenting the indexing and searching for mathematical terms, we have shown the path of the mathematics from the input documents to the user. Next, we will take a look at the concepts of idioms and mathematics together in a discussion about indexing applicable theorem idioms.

3.7 Modelling Mathematics with Idioms

In sections 3.1 - 3.3, I have defined idioms, discussed the theory behind idiom search and presented concrete methods for information extracting by idiom spotting. In sections 3.4 - 3.6, I presented the theoretical background behind the semantics, the indexing and the search for mathematical terms. I will now present how mathematics and idioms fit together to create models for theorem formulations.

When joining natural language idioms and mathematical terms for the purpose of creating an applicable theorem system, it is important to choose the right model for the combined patterns. An *applicable* theorem searcher is different than just a theorem search engine. Finding an applicable theorem requires first of all having a term to apply a theorem to. And this term will most often be used as a search query by the user. Thus, it's important to index (in the mathematical index) specifically the mathematical term which is most likely to be searched by the user. Given the scope of an idiom, we define this as the *desirable term*. The choice of idiom patterns as models to single out *desirable terms* is not trivial, as we can never be sure of the syntax that scientists write theorems in.

The first way to approach the problem of capturing the *desirable term* is to look at theorems which come in the standard $H \Rightarrow C$ form. In terms of natural language idioms, the $\Rightarrow$ symbol can stand for *"implies"*, *"then"*, *"proves that"*, etc. A example of a natural language idiom which fits the $H \Rightarrow C$ model would be:

> **If** $x = \log_a y$, **then** $a^x = y$.

Then, the *desirable term* is found in the hypothesis part and, thus, we must add the math term in the hypothesis ($x \in \mathbb{C}$) to the index. In this case, when the user searches for $x = \log_5 e$ (an instance of H), she will find the theorem which leads her to conclude $5^x = e$ (an instance of C). Thus, it looks like it would be valuable to index only H, and not C.

We may also find theorems of the form $\forall x.\ H \Rightarrow A < B$, where $A < B$ is a complex conclusion (and we can generalize the $<$ symbol to mean any binary operator). A natural language idiom example for this model would be the triangle inequality:

> **For all** $x, y, z \in \mathbb{M}$, **if** d is a distance in the metric space M, **then** $d(x, z) \leq d(x, y) + d(y, z)$.

In a situation where we want to find out $? < d(\vec{a}, \vec{b}) + d(\vec{b}, \vec{c})$ (some lower bound for an instance of the term B), we would search for $@1 \leq d(\vec{a}, \vec{b}) + d(\vec{b}, \vec{c})$ and use the triangle inequality to determine $@1 \rightarrow d(\vec{a}, \vec{c})$. Note that we only need H (in our case *"d is a distance in a metric space"*) to determine the applicability of the theorem here. In this case, the *desirable term* $A < B$ is in the conclusion part, which should be indexed and not in the hypothesis.

Yet another possible model is given as a mixture of the two patterns mentioned above: $\forall f, g.\ H \Rightarrow (A \Rightarrow B)$, where H is a (set of) hypothesis that does not need to be indexed and $A \Rightarrow B$ is a complex conclusion. A natural language example for this model is given by a different formulation of our running example from Figure 1.2:

> **Let** $a \geq 1, b > 1$ be constants, $f(n)$ be a function and $T(n)$ be defined by the recurrent relation
> $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
> Case 1: **If** $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, **then**
> $$T(n) = \Theta\left(n^{\log_b a}\right)$$

In this case, we can consider the *desirable term* to be the complex conclusion $(A \Rightarrow B)$ of *"Case 1"* and we might query for either $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ or for the second term $T(n) = \Theta\left(n^{\log_b a}\right)$, depending on the situation:

- in a situation when we want to reason forward and are stuck with an instance A', we can query to find $A' \Rightarrow X$. For example, a query for $\mathcal{O}\left(n^{\log_2(5)-\epsilon}\right)$
- in a situation when we want to reason backwards and are stuck with an instance B', we can query to find $X \Rightarrow B'$. For example, a query for $\Theta\left(n^{\log_2 7}\right)$

Here, a unification query might also be useful, as might want X to be of a certain form. Given the query possibilities above, it might seem like it is not valuable to index H, but only the complex conclusion. This is, of course, flawed considering that, in this example, H contains the reccurence expression $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, which has been shown as a very likely query throughout this chapter. Thus, we can conclude that the *desirable term* for more advanced models can not be singled out, making the choice between indexing one part and leaving out another undecidable.

Given the theoretical examples presented above, it seems to depend on the theorem idiom type whether the math in the hypothesis or the conclusion should be indexed. My initial approach was to start off with indexing only the conclusion. Then, after seeing more examples from the corpus, I realized that indexing both the hypothesis and the conclusion gives more added value and a higher number of idioms indexed.

A proper complete parsing of formulations like the second and third model given above is still not fully possible with the current version of the Idiom Spotter. The Master Theorem running example fits into the more complex third example pattern above, but is parsed as a simple *if H then C* idiom. Identifying such complex patterns is one of the points for future development of the system. The transition from natural language to mathematical models like $A \Rightarrow (B \Rightarrow C)$ can be achieved either with complex patterns or with a deeper syntactic analysis of the text.

Chapter 4

Idiom Spotter

The previous chapter presented all the theory required for the understanding of idioms and the methods employed for extracting them from input documents. We have also seen a description on how to process the information contained inside the idioms. This chapter will present a description of the Idiom Spotter system, an application which implements the Heuristic Pattern Matching approach for idiom spotting introduced in Section 3.3.2 on XML documents as a proof of concept. I will first give a general description of the application, then describe the algorithms employed and then outline the main components of the system. The final two sections will evaluate the empirical results obtained and mention challenges encountered in the development of the system.

The Idiom Spotter application is released under the Gnu General Public License [Fre91] and its source code can be found in the KWARC repositories, at `https://svn.kwarc.info/repos/sanca/Documents/IUB/Smart%20Systems/Fourth%20Semester/masters/project/idiom_spotting/heuristic`.

4.1 Description

The theory about idiom spotting described in Chapter 3 is implemented in the Idiom Spotter system. This utility, first introduced in [AP09] by Răzvan Paşcanu and the author of this thesis, applies pattern-matching rules to XML documents containing both text and mathematical formulae, in order to extract semantic information about the idioms within. The Idiom Spotter should be viewed as a proof of concept with limitations for the theory about idioms introduced previously.

The main purpose of the Idiom Spotter is to create a database of structured knowledge, aggregating information according to predefined categories. For example, it can create a database of all *theorem idioms* or all *definition idioms* found in the physics papers in the ArXMLiv, a resource which would be extremely useful to any physicist. Storing this information in a queryable format offers information retrieval possibilities later. Thus, in connection with the MathWebSearch system (described in Chapter 5), it forms an applicable theorem searcher, which offers support for mathematical subterm query. Also, by analyzing the frequency of occurrence and the structure of retrieved idioms, we hope to obtain some statistics about the format and draw some conclusions about the most oftenly used constructions found in scientific documents.

The Idiom Spotter system uses the *Heuristic Pattern Matching* approach to extracting information from XML documents. This approach was presented in Section 3.3.2 and is adapted to matching XML texts with mathematical content in any type of representation. The current implementation supports idioms with one or two placeholders per pattern. The extracted idioms and links to their location are stored in `MySQL` databases. The main algorithms employed by the Idiom Spotter system will be explained in detail in the following section.

4.2 Algorithms

The Idiom Spotter takes as input an XML document (or a list of documents) and outputs (to the database and/or standard output) the extracted idioms and their components. The general operation of the Idiom Spotter is as follows:

1. Take in a **document** in XML format as input
2. Parse the **document** into a DOM **tree**
3. Walk the **tree** looking for text **nodes** with idiom *keywords* inside[1]
4. For each text **node** containing an idiom keyword
    - (a) Extract the **scope** of the idiom
    - (b) Split the contents of the **scope** node into **sentences**
    - (c) For each **sentence**
        - i. Compare the **sentence** against available **idiom** patterns
        - ii. If the **sentence** matches an **idiom** pattern
            - A. Extract **components** from sentence according to pattern
            - B. Store **idiom**, **components**, **location** in database
            - C. Output **idiom**, **components**, **location** to output stream

[1] keyword match is case-insensitive

At step 4a of the algorithm listed above, the reason why we take the parent of the current node is that the entire idiom scope is most often larger than just that node. This is particularly the case when we deal with the representation of mathematics in XML documents, where any mathematical term will be represented as a separate `<math>` tag, in MathML or `<OMOBJ>` in OpenMath, breaking up a natural language sentence into several text and math DOM nodes. This means that the mathematical term and its corresponding node in the DOM tree will fall outside of the text node analyzed at 4.

For example, consider the running example in Figure 1.2. A fragment of its representation in an XHTML document (with MathML used for mathematics) is found in Appendix A.2. As we can see, the math node is a separate XML tag than the text node containing the keyword **if**, and the entire sentence spans several tags. The whole idiom **scope** is composed of the ten sibling DOM nodes "*Case 1: If given a recurrence relation of the form*", "$T(n) = aT\left(\frac{n}{b}\right) + f(n)$", "*with*", "$a \geq 1,\ b > 1$", "*and*", "$f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$", "*for some constant*", "$\epsilon > 0$", "*then*" and "$T(n) = \Theta\left(n^{\log_b a}\right)$". Considering that all text in XML documents is implicitly considered to be enclosed in `<text>` tags, an idiom sentence with mathematics often takes up the following form:

Listing 4.1: A possible representation of an idiom in an XML document

```
<p>
        <<text>><<math>><<text>><<math>><<text>>
</p>
```

In the listing above, we use double tags (e.g. `<<math>>`) to show that the nodes are not empty, but are abstracted away from in this representation. As we can see above, the paragraph (which is often the scope of the idiom) contains a list of text and math nodes. In order to "close" a sentence, we must include all the internal nodes from the beginning of the sentence until the ending ".". This is performed at step 4b of the algorithm, where the sentences are split according to the full stop "." symbol.

Sometimes, going to the parent of the **node** containing the keyword is not enough. Depending on the XML representation, the mathematical formula might be situated on a different line, created by introducing (at least) another DOM node level in between the text and the math. For example, in a TEX.XML representation (intermediate LaTeXML output, used for semantic extractions inside the arXMLiv project), you might find the possible example:

Listing 4.2: A possible representation of an idiom spread across different node depths in an XML document

```
<p>
        <text>Thus, we conclude that:</text>
</p>
<equation>
        <math/>
</equation>
```

Here, the father DOM node of the text node containing the keyword **conclude** is the paragraph node `<p>`. It is not the scope of the entire idiom, which would be the combination of both the text and the following `<math>` node (which is not shown fully in the listing above in order to save space). The `<math>` node is actually "hidden" inside an `<equation>` tag which will visually display it on another line. There can be different ways to solve this problem. The Idiom Spotter's approach to it is to look at the father of the text node (in this case the `<p>` node) and add all of its sibling nodes that contain math to the scope, before splitting into sentences. In the listing above, the first sibling (the `<equation>` node) contains math, without which the idiom will not be complete. This approach might lead to adding more nodes than needed for the current **scope** and thus getting more sentences to pattern match. However, since we are in the end only analyzing individual sentences, it does not harm the idiom extraction process, just makes the whole document analysis take longer.

The pattern matching performed at step 4(c)i of the algorithm is done based on the patterns stored in the database. The approach taken here is to consider the serialized string that is the content of the **scope** node, thus mixing together text and (possible) math nodes into one string sentence. Then, this sentence is split into an ordered list of words (or components). Any idiom can also be represented as an ordered list of its keywords and placeholders. Thus, we can compare the two ordered lists for obtaining a match.

Assume Σ is a sentence of length n words. Then $\Sigma = \langle w_1, w_2, ..., w_n \rangle$, and we define an order $\prec$ on sentence words s.t. $w_1 \prec_\Sigma w_2 \prec_\Sigma ... \prec_\Sigma w_n$. Similarly, $\iota = \langle k_1, k_2, ..., k_m \rangle$ is an idiom pattern of length m with $k_1 \prec_\iota k_2 \prec_\iota ... \prec_\iota k_m$. We say that the sentence Σ matches the idiom pattern ι if and only if $\forall i, 1 \leq i \leq m, \exists j\ s.t.\ k_i = w_j$ and $\forall i, 1 \leq i < mk_i \prec_\Sigma k_{i+1}$. Thus, a sentence matches an idiom pattern, if the ordered list of idiom words is an ordered subset of the sentence list of words.

The algorithms explained in this section are performed by the main Idiom Spotter script, which is the core component of the system. The Idiom Spotter architecture is quite simple and is explained in the next section.

4.3 System Architecture

The Idiom Spotter system has a very simple architecture, due to the fact that it does not perform any inter-process communication or user interaction. The Idiom Spotter is an application which takes in XML documents as input, runs idiom extraction on them and saves the results to a database. Thus, it is designed as one main component, which only communicates to a database, which keeps all the idiom-related information.

The Idiom Spotter creates and updates a MySQL database it uses for its operation.

Before any idioms can be spotted, the database needs to be created, holding all the information needed for idiom retrieval and providing a storage space for extracted items. Special tables in the database hold all the idiom patterns and the main keywords that represent them. These are loaded into memory at extraction time and matched against the input documents. An *idioms* table is the storage space for all idioms, with their entire scope node and an XPath pointing to this node, identifying their location within a document. The other tables in the database store the placeholder types and differ according to idiom category, and the corresponding number of placeholder types.

The current version of the Idiom Spotter provides support for idioms with two placeholder types. In this case, the Idiom Spotter database contains a table for each of the types, with pointers to the idiom in the *idioms* table that they are part of, and pointers to connect them in between each other, in order to model *one-to-many* or *many-to-many* relationships. As mentioned in Section 3.3.2, the semantics of some idioms allows the placeholders to be connected, as this provides more knowledge. For example, for the *theorem idiom* category, a *conclusion* will always point to at least one *hypothesis*, and several *hypotheses* might point to the same *conclusion*.

After the Idiom Spotter is run on the target corpus, the user can check the results by looking directly into the MySQL database. A view of the idiom spotting results in the *phpMyAdmin* utility interface is shown in Figure 4.1. There, we can see the *idioms* table of an Idiom Spotter designed to find theorem idioms after a run on the Connexions repository.

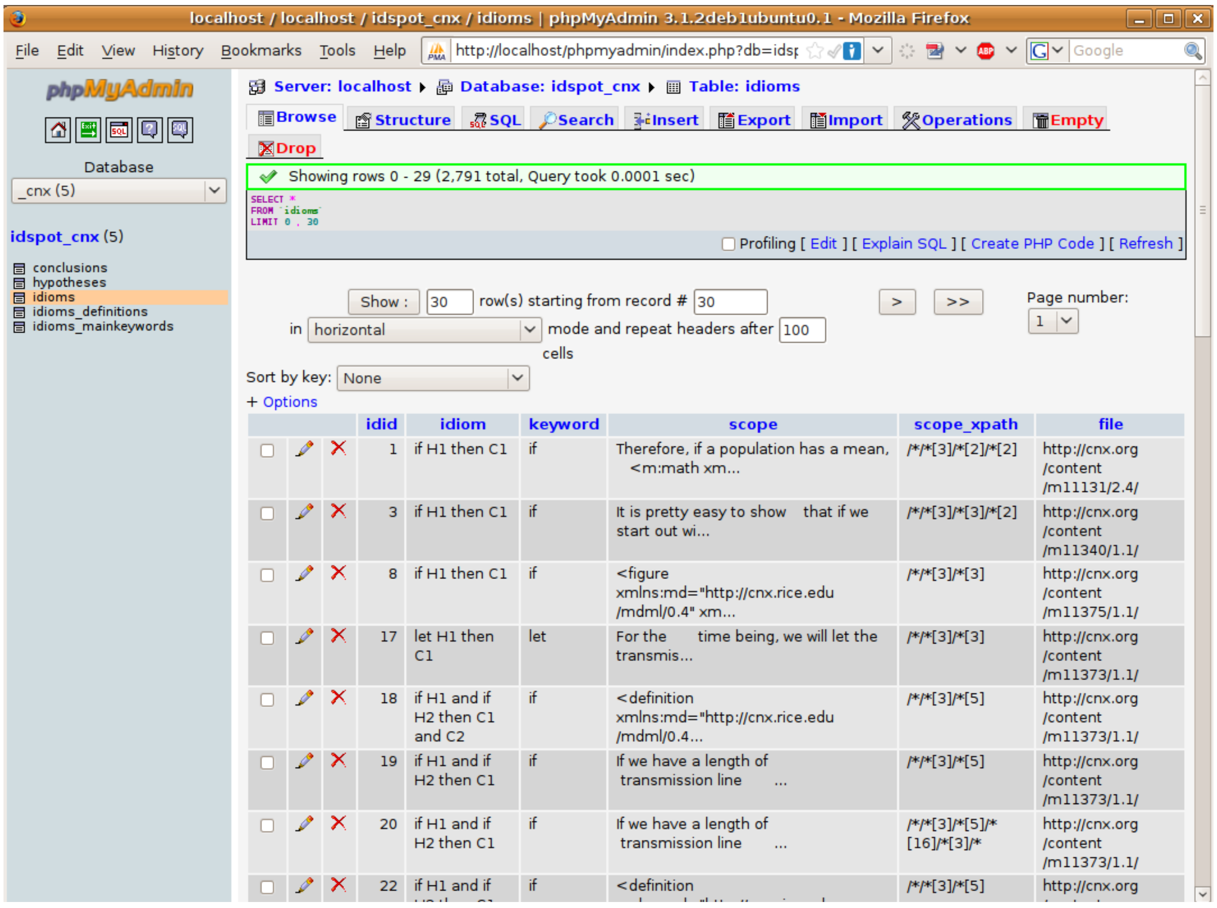
Having explained the basic architecture of the system, let us now analyze the tests which were performed with the Idiom Spotter, in order to evaluate its performance and the validity of the theoretical assumptions made in Chapter 3.

4.4 Performance and Empirical Results

In order to test the performance of the system with respect to the number of theorems extracted, I have run several tests on various corpora, described in the following section. The test results are given as well, alongside an initial evaluation. Finally, I will give a note about the performance of the system.

4.4.1 Test Corpora

In order to extract any meaningful statistics or for the search functionality to make sense, we need to run the Idiom Spotter on a large collection of documents. Thus, the system was tested on several corpora, each providing different field orientations. The four corpora used as testcases are described below:


localhost / localhost / idspot_cnx / idioms | phpMyAdmin 3.1.2deb1ubuntu0.1 - Mozilla Firefox

http://localhost/phpmyadmin/index.php?db=idsp

Server: localhost ▸ Database: idspot_cnx ▸ Table: idioms

Showing rows 0 - 29 (2,791 total, Query took 0.0001 sec)

```
SELECT *
FROM `idioms`
LIMIT 0 , 30
```

Show : 30 row(s) starting from record # 30

in horizontal mode and repeat headers after 100 cells

Page number: 1

Sort by key: None

			idid	idiom	keyword	scope	scope_xpath	file
			1	if H1 then C1	if	Therefore, if a population has a mean, <m:math xm...	/*/*[3]/*[2]/*[2]	http://cnx.org/content/m11131/2.4/
			3	if H1 then C1	if	It is pretty easy to show that if we start out wi...	/*/*[3]/*[3]/*[2]	http://cnx.org/content/m11340/1.1/
			8	if H1 then C1	if	<figure xmlns:md="http://cnx.rice.edu/mdml/0.4" xm...	/*/*[3]/*[3]	http://cnx.org/content/m11375/1.1/
			17	let H1 then C1	let	For the time being, we will let the transmis...	/*/*[3]/*[3]	http://cnx.org/content/m11373/1.1/
			18	if H1 and if H2 then C1 and C2	if	<definition xmlns:md="http://cnx.rice.edu/mdml/0.4...	/*/*[3]/*[5]	http://cnx.org/content/m11373/1.1/
			19	if H1 and if H2 then C1	if	If we have a length of transmission line ...	/*/*[3]/*[5]	http://cnx.org/content/m11373/1.1/
			20	if H1 and if H2 then C1	if	If we have a length of transmission line ...	/*/*[3]/*[5]/*[16]/*[3]/*	http://cnx.org/content/m11373/1.1/
			22	if H1 and if	if	<definition	/*/*[3]/*[5]	http://cnx.org

Figure 4.1: A view of the IDIOM SPOTTER *idioms* table after a run on the Connexions repository in phpMyAdmin

1. The **LaMaPUn group sandbox** (available at: `http://arxmliv.kwarc.info/files/sandbox/nlp-sandbox.tar.gz`) This "sandbox" is a finely picked selection of documents with mathematical content, as it was intended for *mathematical formula disambiguation*. As the first testcase for the LaMaPUn architecture (see [GJA+09], Section 2.1.4), its intended use is to help develop and test tools for LaTeXML output *purification mathematical formula disambiguation*. Characteristics:

   - approximately 2200 `.tex.xml` papers
   - focused on basic mathematics only, no physics articles (with usually more complex math formulae)
   - all error-free .xhtml papers from the transformed ArXMLiv corpus that are not physics related
   - mathematics in XMath representation is likely to be incorrect

- used for testing in May 2009

2. The **arXMLiv Web Development sandbox**(WebDev) (available at `http://arxmliv.kwarc.info/files/sandbox/webdev-sandbox.tar.gz`). As part of the ArXMLiv project (see [SK08], Section 2.1.2), this collection of documents (called "sandbox") is a selected subset of all the files transformed from LaTeX into XHTML, without errors. Its intended use is to test out document indexing and annotation tools. As such it has the following characteristics:

   - approximately 5000 error-free converted `.tex.xml` documents
   - all images and extra file-weight stripped off
   - mathematics in XMath representation is likely to be incorrect
   - used for testing in May 2009

3. The **Connexions database** available under `http://cnx.org/content`. The Connexions system [CNX09] is an online platform for publishing user content. It is oftenly used to publish engineering courses and thus contains documents with mathematical formulae. The Connexions repository is indexed by MathWebSearch. Characteristics:

   - approximately 12200 `.cnxml` source pages (pure content, without display elements)
   - documents not restricted to Mathematics
   - all (supposedly) correct MathML
   - non-standard XML format
   - used for testing in August 2009

4. The **Saarbrücken sandbox** (available at `http://arxmliv.kwarc.info/files/sandbox/saarb-sandbox.tar.gz`). As further research on the arXMLiv project, this "sandbox" was created in the summer 2009 as part of a research effort by Deyan Ginev, under the supervision of Magdalena Wolska at the Computational Linguistics and Phonetics department of Saarland University. It contains only Mathematics articles and its intended use is for linguistic processing. As such it has the following characteristics:

   - approximately 10200 `.tex.xml` documents
   - all from the field of Mathematics
   - contain an abstract
   - at least two `theorem` blocks per document
   - mathematics in XMath representation is likely to be incorrect

- used for testing in August 2009

The documents with a `.tex.xml` extension (intermediary LaTeXML conversion step) from the arXMLiv corpus contain a representation linguistically equivalent to the LaTeX source document. The mathematical formulas are represented in an XML format called XMath [xma08], which is an intermediate format from which MathML is created. The LaTeXML conversion step from LaTeX to XMath introduces a lot of (mostly flawed) assumptions in the attempt to infer content semantics from the visual presentation. XMath is currently not indexable by MathWebSearch.

Testing on these corpora was done with a small group of idioms, presented below in Table 4.2 containg mostly theorem idiom patterns. Only the idioms containing mathematical terms (in MathML or XMath) were indexed. Apart from the LaMaPUn and the Saarbrücken sandboxes, the documents in the other two databases are not restricted to mathematics or text with mathematical formulae. While it is likely that most scientific articles in the arXMLiv repository contain some mathematical terms, the Connexions repository is open to all sort of content, and thus a smaller fraction of the documents are expected to have mathematics. Also, it is worth mentioning that the documents from sources 2, 1 and 4 are automatically converted to XML from LaTeX. Thus, the mathematical terms within have a semantically questionable and quite often flawed representation (for more details, see Section 7.1). The mathematics in the Connexions repository is all written by human authors, in Presentation or Content MathML, and thus assumed to be formally correct. However, as all human input is subjected to mistake, there are documents where the MathML is formally incorrect. This does not make a difference for the idiom spotting process but will make a difference between indexable and non-indexable documents in the Applicable Theorem Search system later(see Chapter 6). The sources of the documents in the Connexions repository are stored in a custom CNXML format, which contains only the contents of the files, without visual markup. This provides a cleaner XML. The test results are only quantitative, and not qualitative, so we can not yet estimate the number of false positives or idioms which were left behind. What we look at is the total number of idioms found and the percentage of files that contain idioms in the corpus.

4.4.2 Results

The Idiom Spotter was run on the repositories described above, creating four different idiom databases. The general results of the tests are presented in Table 4.1. More detailed results, presenting all the idiom patterns tested and the number of occurences in each corpus are seen in Table 4.2.

As we can see in Table 4.1, our first approach at spotting idioms finds very different results across the different corpora tested on. Starting from very few idioms on the Connexions corpus, going up to an average number of idioms for the LaMaPUn and WebDev corpus

Corpus	**LaMaPUn**	**WebDev**	**Connexions**	**Saarbrücken**
Total files	2266	5006	11712	10239
Files with idioms	1080	2142	451	9947
Files with idioms (%)	47%	42%	3.8%	97%
Idioms found	**5906**	**7334**	**1794**	**215044**
Average idioms per file (total)	2.6	1.46	0.15	21
Average idioms per file (only files with idioms)	5.46	3.42	3.97	21.6

Table 4.1: Corpora Statistics

and arriving at a huge number of idioms on the Saarbrücken corpus, the results are clearly influenced by the document profile of the corpora.

The almost perfect percentage of files that have idioms in the Saarbrücken corpus (97%) seems to be highly influenced by the way this sandbox was created (mathematical papers with at least two theorems). The average of 21 idioms per file seems to confirm our initial assumptions made in Section 3.1 about the use of idiom structures in mathematical texts. Actually, the file with the highest number of idiom occurences contains over 500 matches, and discusses a topic in Group Theory (*"Regular Neighbourhoods and Canonical Decompositions for Groups"*). We assume such papers provide classical proof steps or theorem lists using the standard formulation encoded in the idiom patterns. At the other extreme of the spectrum lies the Connexions repository, with more than 100 times less found idioms in a corpus of similar (actually, slightly larger) size. The very low (3.8%) percentage of files containing idioms in the Connexions repository shows that the intersection between the documents containing mathematics and our idiom patterns is quite low. Unfortunately so, we had expected a result in this direction, given the open nature of the platform to any user content.

The LaMaPUn and WebDev corpora share a similar percentage, which is around 40%. This is not a low rate and it seems to be an indication that the documents inside these databases do not have condensed mathematical content. Although the LaMaPUn sandbox is picked only from mathematics papers, it does not have a strict heuristic in terms of the mathematical content, like the Saarbrücken corpus. This leads us to believe that a lower fraction of the documents in this sandbox contain theorems, as opposed to the other, larger, theorem oriented corpus. The experiment of running the IDIOM SPOTTER on these corpora was not aimed at extracting all possible idioms, but at proving that our first approach at applying the presented theoretical background behind idiom spotting gives us a positive start and some more knowledge about idioms in scientific texts.

In Table 4.2, we can see all of the tested idiom patterns and their frequency of occurence in the four corpora. This table once again shows the demonstrative nature of these tests, trying out only 13 idiom patters (11 of which are theorem idioms). The interesting things to

Idiom pattern	**Frequency LaMaPUn**	**Frequency WebDev**	**Frequency Connexions**	**Frequency Saarbrücken**
assume H1 then C1	55	105	29	1755
conclude D1 is D2	100	217	22	3176
define D1 to be D2	157	152	58	4911
given H1 then C1	77	117	43	1809
H1 if and only if C1	502	392	56	25979
H1 implies C1	801	1545	170	30593
C1 only if H1	694	782	102	27964
C1 only when H1	146	290	35	1553
if H1 and if H2 then C1	58	65	24	2165
if H1 and if H2 then C1 and C2	40	47	10	1326
if H1 then C1	**3098**	**3438**	**1161**	**105142**
let H1 then C1	142	140	61	6915
suppose H1 then C1	36	44	23	1756
Theorem patterns	**5649**	**6965**	**1714**	**206957**

Table 4.2: Statistical results of the Idiom Spotter on different corpora

observe are the idiom patterns that lie at the extremes. The *"if H1 then C1"* pattern seems to be the most frequent in all three corpora with over 110000 occurences in total. The *"if H1 and if H2 then C1 and C2"* and the *"suppose H1 then C1"* are the least frequent. This is expected from the former, since it is a specific case of the *"if H then C"* idiom, but might show that the *"suppose"* pattern is less fequently used in scientific text. This idiom has only 1859 occurences, ranking lowest in two out of four corpora. What is interesting to note here is that in the theorem-dense Saarbrücken corpus, *"suppose H1 then C1"* is only 4th last. This could be an indication that this pattern is more frequently used in mathematical theorems and proofs, rather than in other scientific fields. Another interesting statistic is that of the higher number of *"H1 implies C1"* idioms in the Web Development "sandbox" (not comparing to the Saarbrücken sandbox). This can be interpreted as an indication that authors tend to stick to classical logical argumentative proofs for their statements in scientific articles in all fields.

With the lowest number of theorem idioms of all the four corpora (1714), the Connexions repository seems to be a corpus of relatively low use for the Applicable Theorem Search system. This is somewhat natural, since all the other corpora are made up of published scientific articles, while the Connexions corpus only contains a relatively low number of online courses. However, for the reason that the mathematical content in the arXMLiv repository is not yet represented in a correct semantic format, we are constrained to index only the Connexions repository. More details are given in Chapter 7.

Finally, we present Figure 4.2, which shows the distribution of idioms per file in the Saarbrücken corpus. The y axis represents the number of files and the x axis the number of idioms per file. As it could have been expected for such a small list of patterns, there are a lot of files with just a few idioms (the peak is at 383 files with only 6 idioms). However, the curve does not drop down too fast, and it only goes below 100 files at more than 33 idioms per file, which is above the average 21, seen in Table 4.2. There are also files with hundreds of idioms per file. At over 100 idioms per file, the file count drops below 7. The extreme at the other end is a file with 527 idioms, as mentioned above. This gives us clear evidence that there are idiom-rich files in our repositories. It is very likely that the corpus contains even more idioms, we just have to improve the idiom extraction methods (whether that means more complex patterns or more simple patterns) in order to get as many of them as possible. The Saarbrücken corpus seems to be a great "sandbox" for testing the Idiom Spotter system and an ideal database for the Applicable Theorem Search system. However, problems related to the representation of the mathematical terms within prevent us from being able to index the theorems inside.

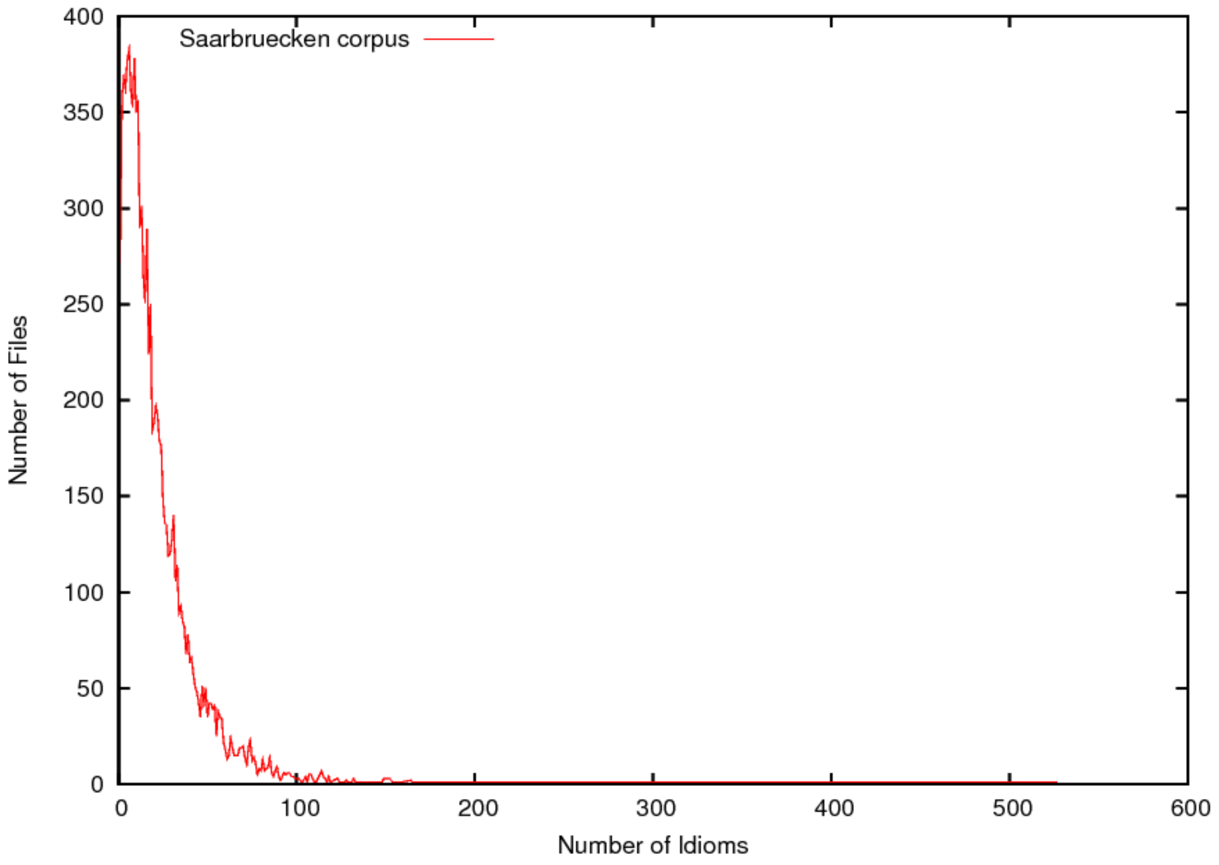


Figure 4.2: The idiom distribution per file in the Saarbrücken sandbox

Note: The tables and figures above show results of the initial implementation of the Heuristic Pattern Matching approach for idiom spotting on documents in XML format.

It is possible that a lot of theoretically matching idioms are not harvested because of the nature of the XML representation of mixed text and mathematics discourse and the difficulties which appear in processing it. Also, due to the linguistic flexibility of natural language, it is possible that the system captures false positives. For details about planned future improvements of the Idiom Spotter, please see Chapter 8.

4.4.3 Performance

The Idiom Spotter perfomance is not crucial in terms of time, as it provides a service which only needs to be run periodically on the input corpora. It only needs up to one second to parse and store all the idioms out of a file with high idiom content. A run on the Saarbrücken corpus with condensed idiom content takes around four hours to complete. It has a memory footprint of around 50MB, since it never stores more than one document in memory at the same time. For future development, improving performance is not considerd a priority, as opposed to improved recall rate.

4.5 Implementation Challenges

Together with Răzvan Paşcanu, I have implemented a prototype version of the Idiom Spotter in Fall 2008 as part of the Computational Semantics Laboratory held by Prof. Dr. Michael Kohlhase at Jacobs University Bremen [AP09]. The implementation was done in Python, using bindings for the `libxml2` and `MySQL` libraries. I continued the development of the Idiom Spotter and its integration with the MathWebSearch system in Spring 2009.

While the Heuristic Pattern Matching approach to idiom spotting is rather simple in theory, its implementation in the Idiom Spotter system came across quite a few problems. I have gathered these problems and some pitfalls of my implementation approach in this section, as a legacy of my experience and advice for future researchers attempting a similar project.

- **Separating placeholders** While there can be two keywords, one following the other in an idiom pattern, it does not make sense to have placeholders one after the other. For example, *For all H, C* is such a pattern. The main reason is that in such a situation the Idiom Spotter can not decide where the first placeholder ends and where the second one begins. Note though that having a comma *","* as a keyword separating placeholders can be misleading and error-prone, as commas occur frequently in informal discourse. For example, in the idiom *"For all $x \in \mathbb{N}$, $y \in \mathbb{Z}$ and $z \in \mathbb{Q}$, $xy \in \mathbb{Z}$, $xz \in \mathbb{Q}$ and $zy \in \mathbb{Q}$"* we have three commas appearing in the middle of the mathematical terms, but only the second one divides the hypothesis from the conclusion. It is impossible to decide strictly from the syntax of the formula, which comma is a keyword.

- **Retrieving files from the repository** The sources of the online documents are not always easily retrievable. For example, the Connexions repository (`http://cnx.org/content/OAI`) is an OAI repository which does not expose a file server, but responds to user queries about content files. Thus, in order to download the documents in the right format, the right query must be sent. When trying to download the files with the linux application `wget`, we must request the *application/xhtml+xml* mime type from the server. Otherwise, the documents are returned as a *file/html* mime with no Content MathML inside. For this, the header of the request must be modified with a parameter of the type `--header='Accept: application/xhtml+xml'`.

- **Special XML format** The Connexions module sources are written in a custom XML format called CNXML. This is an *XHTML Transitional* format and needs special DTDs to be loaded for validation. CNXML 0.5 uses DTDs while CNXML 0.6 uses RelaxNG for validation. At first, I tried to validate the CNXML documents but that failed due to problems with the `libxml2` parser not being able to find the DTDs on its own. In the end, I changed the Idiom Spotter to run a parser with special flags set **not** to validate the documents. This was enough to build a DOM tree and perform pattern matching.

- **XML Entities** The CNXML format uses special XML entities in Content MathML nodes in order to specify mathematical symbols (e.g. `&theta;` for θ, `&sigma;` for σ). It turned out that most documents containing MathML were using such entities. Parsing a document containing these entities fails unless they are added by a DTD or catalog of DTDs. Since the parser had problems retrieving the DTDs from the internet, I had to add them locally to the machine on which the parser was running, by installing special `Debian` packages, provided by Connexions. This solved most problems but some documents still contained unknown symbols and were thrown out by the `libxml2` parser.

- **Input encoding** The Connexions platform provides open input opportunities for authoring online documents. Thus, authors use a variety of operating systems when inputting text and characters are sometimes entered in different encodings. The Idiom Spotter opens only those documents which are in UTF-8 encoding. All others (in latin-1 encoding for example) are skipped.

The high number of theorem idiom patterns tested and the matched results of the spotting process on the given corpora presented in this chapter motivates the author's choice of indexing theorem idioms for search. The Applicable Theorem Search *system, presented in Chapter 6 provides indexing and search facilities for theorem idioms by their mathematical content. The mathematical subterm indexing and search functionality is provided by the* MathWebSearch *system (presented in the following chapter), while the idiom spotting and storing facilities are provided by the* Idiom Spotter. *The* Applicable Theorem

SEARCH *system joins and interconnects the two into a combined system for idiom retrieval and presentation.*

Chapter 5

MathWebSearch

After presenting the Idiom Spotter in the previous chapter, we have the pratical means for finding theorem idioms. The next thing that we need for the completion of the proposed Applicable Theorem Search system is a search engine for indexing the mathematical formulas found within. As mentioned before, we use the MathWebSearch system for this task, which we will analyze in the following sections. This chapter provides a thorough description of the MathWebSearch system architecture as a basis for its extension and as a motivation for the design decisions taken to build the Applicable Theorem Search system.

The MathWebSearch system is released under the Gnu General Public License [Fre91] and was created inside the KWARC research group at Jacobs University. Its source code can be found in the *mathweb.org* repository, at `https://svn.mathweb.org/repos/mws`.

5.1 Description

The MathWebSearch(MWS) system [Mat09], as described in 2.1.1, is a search engine for mathematical terms in XML documents. It offers a service of subterm indexing (and querying) to the user, bringing semantic mathematics search in the information retrieval world. MathWebSearch can process any XML-based content representation of mathematical formulas: MathML [ABC+03] and OpenMath [BCC+04] are supported directly, other formats e.g. Wolfram Research's Mathematica [Wol99] are supported if a converter for them is provided.

Consider the example in Figure 5.1: We have the standard mathematical notation of an equation (1), its Content MathML representation (2), and the term we extract for indexing (3). Note that in the internal (MathML Query) syntax, the query variables and the generalization targets in the index are represented with identifiers starting with the `@` character. See [Mat09] for more details and syntax of search queries.

(1) Mathematical expression:
$f(x) = y$

(2) Content MathML:

```
<math>
 <apply><eq/>
  <apply>
    <ci>f</ci>
    <ci>x</ci>
  </apply>
  <ci>y</ci>
 </apply>
</math>
```

(3) String representation:
$eq(@f(@x), @y)$

Figure 5.1: Converting to MathML Query

Mathematical terms like the one in the figure are not stored in their original form in the index. MathWebSearch 0.4 stores formulas with universal variables in their generalized form. In the example in Figure 5.1, we identify f, x and y as universals (defined in Section 3.4) and store the formula in generalized representation in the index. Thus, the above formula is stored, for example, as *@1(@2)=@3*, where the @ variables are universal and can be later instantiated with a term at search time.

This form of representation in the index allows for different types of user queries, all stemming from term unification:

1. instantiation queries: A search for $\int_{@2}^{@1} s^2(t)dt$ finds

$$\frac{1}{T}\int_0^T s^2(t)dt = \sum_{k=-\infty}^{\infty} \|c_k\|^2$$

   with substitutions $@1 \rightarrow T$ and $@2 \rightarrow 0$.

2. variation queries (α-renaming): A search for $\sum_{i=1}^{n} i$ finds $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$ with substitution $i \rightarrow k$.

3. generalization queries: A search for $\frac{1}{\pi}\int_0^{\pi}(f+g)^2(t)dt$ matches the index term

$$\frac{1}{@1}\int_0^{@1} (@2)^2(@3)d@3 = \sum_{k=-\infty}^{\infty} \|@4_k\|^2$$

   with substitutions $@1 \rightarrow \pi$, $@2 \rightarrow (f+g)$, $@3 \rightarrow t$.

4. unification queries: A search for $@1\int_0^{\pi}(f+g)^{@2}(t)dt$ matches the term

$$\frac{1}{@3}\int_0^{@3} (@4)^2(@5)d@5 = \sum_{k=-\infty}^{\infty} \|@6_k\|^2$$

   with substitutions $@1 \rightarrow \frac{1}{\pi}$, $@2 \rightarrow 2$, $@3 \rightarrow \pi$, $@4 \rightarrow (f+g)$, $@5 \rightarrow t$.

5. and any combination of these.

The theoretical background behind these searches has been given in Section 3.6, but I have listed them again together here for the sake of completeness. In most of the query examples above, note that the actual term which made up the query is only a subterm of the mathematical formula returned. The latter is a self-contained term, represented for e.g. as a `<math>` node in Content MathML (as seen earlier in Figure 5.1). This shows the semantic subterm indexing nature of the MathWebSearch system which, in its crawling and information aggregating stage, not only stores the mathematical terms but also adds all of its subterms to the index. This allows for an increased searching power and simpler, less specific querying requirements on the user. Also, being able to search for mathematical terms by their structure and not by their visual presentation, avoids a lot of implicit ambiguity, as described in [KS06].

The MathWebSearch system is a distributed research project with varying developers. The initial implementation (MathWebSearch 0.1) was made by Ioan Şucan in 2006. Then, I made the MaTeSearch extension to the system in the summer of 2007. In 2008, MathWebSearch 0.4 was launched, having an improved index and the new unification-based query functionality developed by Constantin Jucovschi with my support for integration and universal extraction (by the Variable Spotter). The new web interface for the system was developed in 2008 by Alberto Gonzalez Palomo, along with the integration of Sentido [Pal06] as a query formula editor. For the extension of the math search engine to the Applicable Theorem Search system, I have been close collaboration with all these developers in order to provide a system which integrates seamlessly with MathWebSearch.

The MathWebSearch system is quite complex and encompasses several tasks: crawling, indexing and retrieving the information [KAJ+08], as most three-tiered search engine application suites do. In order to build up on and extend the MathWebSearch system with the functionality needed by the Applicable Theorem Search system, the author was faced with the task of understanding and *reverse-engineering* this search engine and all of its composing parts. In the following section, we will discuss the architecture of the MathWebSearch system, to the minimum level of detail necessary to enable the reader to understand the steps which were taken to extend it and the motivation behind the design decisions made.

5.2 System Architecture

As described in [KAJ+08] and pictured in Figure 5.2, the MathWebSearch server is a distributed application consisting of:

Search Nodes that run search servers, index builders, and web crawlers. Some of the nodes contain “meta-servers” that act as gateways to others; they forward queries to a specified set of nodes and merge the received results. Thus the collection of search nodes is organized as a tree for efficient query distribution when using a large number of nodes. In case of failure of a node, the only effect is that the results that would have been produced by that node are not received by the web server.

Database Servers that store the indexed documents here realized in MySQL.

A Web Server to Communicate with Browser Clients that combines search results from the root meta-server with the documents from the database.

An Admin Server that allows to assign the different tasks to the available nodes, or to add automatic load balancing if needed. The admin server also monitors the search nodes for node failures and re-directs search.

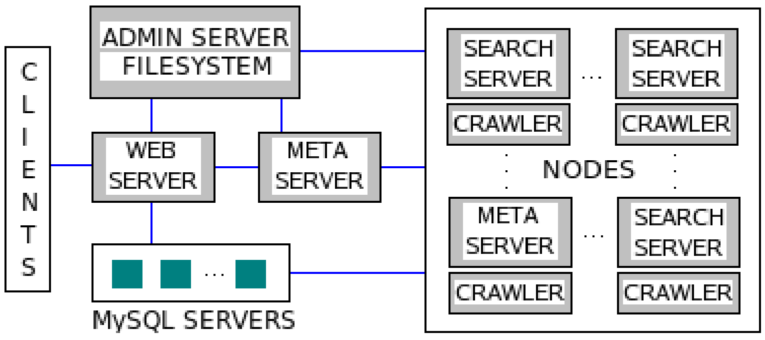


Figure 5.2: MathWebSearch System Architecture [KAJ+08]

The idea of independent Search Nodes reinforces the modularity and universality of the system. Thus, the easiest way to maintain search on different repositories is to create different Search Nodes for them. Also, two repositories with different document representations (e.g. MathML and OpenMath) will naturally have different Search Nodes assigned to them. Following the three-tier approach to search, each Search Node usually implements the same stages:

- **Crawling** a (usually web) repository in order to retrieve the XML documents to be added to the database index. The output of a crawler is usually a list of links or a list of files (already retrieved), waiting to be further processed. This stage is done “offline”, periodically (e.g. monthly, twice a year, etc).

- **Indexing** the relevant information by parsing each of the documents provided by the crawler. For MathWebSearch, the MathML or OpenMath representation of the mathematical terms is the only interesting information. This information is added both to a MySQL database of document links and XML terms, but also to an internal database, accessed by a C++ search server. This stage is performed after every crawl, so also just periodically.

- **Searching** is the final stage and the only one exposing functionality to the user. At this stage, the Search Node must support a Search Server API that handles query requests from the user. It must, in turn, use this information to query the databases and return the results in a particular presentation back to the user. This stage is performed “online”, at run time, so the server must be robust and efficient. Each server is able to run multi-threaded searches, the maximum number of threads being chosen at indexing time.

These three stages must be followed for every new resource (repository) that is added to the system. For this reason, it is easier to manage the source nodes independently, rather than trying to aggregate all data into the same workflow. The aggregation is performed at search time, when the metaserver that communicates with the nodes comes into play. Following, a more detailed description of the architecture is given, in order to motivate the extension of the system to Applicable Theorem Search.

Design Decision and Contribution *In order to use the* MathWebSearch *system as a basis for* Applicable Theorem Search, *the indexing and searching stages must be modified. The extensions to* MathWebSearch *need to provide the capability to index the idiom parts (as string components) alongside the math formulas in the database and keep the connection between each indexed math formula and the idiom (in the Idiom Spotter database) that it is part of. Also, the searching functionality needs to be extended in order to retrieve these extra parts for each result both from the math term database and from the idiom database. I have added this required functionality, with the intention to keep as much of the original* MathWebSearch *code unchanged. The end result was a branch of the code which does not break the basic* MathWebSearch *functionality, provides the required* Applicable Theorem Search *extensions and can coexist and run in parallel with the original system.*

5.2.1 Search Server

The very back-end of the MathWebSearch system is represented by C++ code which implements the *substitution-tree indexing* algorithm described in [KS06] and in Section 3.5. This algorithm performs the *term indexing* referred to above and defined in [Gra96] and stores all the subterms in one *substitution-tree* (see Figure 3.1). This index stores all mathematical terms in an internal string representation. The transformation to this string representation of the math terms is done by crawlers (which will be described below in Section 5.2.3) from the various representation formats that the Math formulas are found in.

The Search Server works as a “black box”, providing simple and fast functionality inside the Search Node (see Figure 5.2). At the time the index is built, a C++ program takes in a list of mathematical terms in string representation and outputs a binary data file which constitutes the *substitution-tree index* mentioned earlier. This index data file contains an integer for every term to identify the **document** from where it comes and another integer for its ordinal **position** in this file (since there is usually more than one mathematical term per indexed file). This operation is part of the indexing stage described above in Section 5.2 and is performed offline and periodically. At runtime, the search server needs to load this file, and keep the entire index in memory, as it awaits queries on its input pipeline. The search server is accessed through a socket on a predefined port and is sent a query in string representation (usually by the Meta Server). This query may contain wild cards like `@2`, which will be matched against all available terms at that position in the tree [KS06]. The server returns a list of integer pairs, each representing the **document** identifier and the **position** identifier in the document for every match found and the MySQL database where they are stored.

Basically, if comparing against the three tiers of search engines introduced above, the Search Server performs both *indexing* and *searching*. The search server has nothing to do with the *crawling* stage, as it does not deal with finding documents on the web. Since the terms are directly provided to it, some part of the indexing is already done, namely the document parsing, term extraction and the term transformation from XML to the string format. Thus, the Search Server itself only performs the mathematical indexing in the substitution-tree at indexing time. At search time, it also performs the matching of the term received as input from the Meta Server against the index and returns all the results it found back to the latter. These results have to be further processed, as they do not carry more than identifiers of the resulting documents.

5.2.2 Meta Server

As mentioned above, there is a Search Server for every Search Node and it usually serves one repository index. The information sent to and received from each Seach Server is disseminated, respectively aggregated by the single Meta Server, which lies outside the conceptual Search Nodes. The Meta Server only communicates with the Web Server, through a socket on a predefined port, and acts as a *funnel* for the data. The Meta Server's main function is to receive the query from the Web Server in string format, forward the same query to each individual Search Server and then aggregate the results that it gets back from them into one set. This set of results is sent back to the Web Server, for further processing. Thus, the Meta Server only takes part in the "on-line" *searching* stage, and must be active continuously at runtime.

Design Decision and Contribution *Considering the fact that the Search Servers only deal with indexing the mathematical formula in internal string representation in the substitution-tree index, I considered it a good idea to leave this unchanged. Storing the string data corresponding to the idioms in the C++ index would have been an unneccessary complication and would have required a lot of interference with the internal representation of the index itself. Considering that the XML representation of the mathematical terms is kept in the MySQL databases anyways, storing the idiom information there seemed to be the most natural choice. Thus, even in the* APPLICABLE THEOREM SEARCH *system, I kept the Search Servers and* *the Meta Server were as a black box, which were I only used for storing and retrieving the information about the math terms at index, respectively search time.*

5.2.3 Crawlers

Repositories of scientific documents often have differing structures. For example, the Wolfram Function repository [Wol09] is an OAI repository, while the arXiv [arX07] repository is a simple file server. Thus, MathWebSearch implements different functionality in its crawlers, being able to retrieve the files from several different types of storage and in different formats. At current time, MathWebSearch has a crawler for OAI repositories, a web crawler that searches for all the links found in the starting document, a local crawler that parses files from the local file system and a file list crawler that downloads and processes all the links in an input list. All these are managed by a *meta-crawler* which takes as input the repository type and the starting point (whether a list of URLs or a single URL which will be used to retrieve a list of documents, depending on the repository) for the crawl. This crawler and all of the other specialized crawlers that it manages (which will from now on be refered to collectively as "the MathWebSearch Crawler") makes up the entire crawling functionality of a Search Node and is pictured in Figure 5.2.

The *meta-crawler* calls the other aforementioned crawlers in turn, if they are needed, until a final list of URLs (whether pointing to files on the Internet or the local filesystem) is obtained. Then, this list is processed by a `FILELISTcrawler` script, which downloads the documents (if needed), analyzes the content, retrieves the mathematical terms and stores them as XML strings in a MySQL database. Then, for each term, a translation is made from XML to the string format that is fed into the substitution-tree index, described earlier. This string, together with the file identifier and the location identifier defined above are stored alongside each XML term in the MySQL database. Then, at the end of the crawl, a separate script (also managed by the *meta-crawler*) pulls all the string math terms out of the database and feeds them to the Search Server utility that creates the binary data math index file.

A bit misleading, the MathWebSearch Crawler performs both the *crawling* and *indexing* offline stages of the search engine. On the one hand, it retrieves all the 'promising' documents from the web, storing them as a list of links, accounting for the *crawling* stage. On the other hand, it carries out the *indexing* stage in two steps:

- In the first step, each of the links is processed in turn, retrieving the math XML terms and their string representation from the scientific documents and storing it in the MySQL database.
- In the second step, once all the links have been processed, all the math terms in string representation with their location information are taken out from the MySQL database and used to create the substitution-tree index.

Thus, the index is shared between the substitution-tree for the math terms and the MySQL database for the document information.

As mentioned in Section 5.1, MathWebSearch needs semantic content representations of mathematical terms in order to add them to the index. However, since repositories of Content MathML or Open Math are still a scarce resource, the MathWebSearch Crawler also indexes Presentation MathML, in an attempt to add as many terms to the index as possible. Little semantics can be inferred from Presentation MathML, so these terms are only retrievable if the user query is entered as Presentation MathML or if the Presentation format of the terms happens to match the Content representation (only for simple terms like $log(x)$, for example).

Design Decision and Contribution *Given the need to extract the idiom components as strings, store them in a database and keep the connection to the math terms found by the* MathWebSearch *Crawler, I decided to extend the first step of the indexing stage described in the above paragraph.*

The crawling *stage need not be modified, as the same documents which are interesting for the* MathWebSearch *system are to be analyzed for idioms by the Idiom Spotter. However, in the indexing part which interfaces with the MySQL database, I have made some changes to synchronize the Idiom Spotter output with the math term extraction. More specifically, I modified the* MathWebSearch *Crawler to update the MySQL database with some information regarding the idioms found at the particular mathematical term. However, as one of my goals was to maintain the basic* MathWebSearch *functionality, I took extra care not to interfere with the indexing of the terms which do not belong to any idiom (they are found in a sentence which does not match any predefined idiom pattern).*

Therefore, I made the decision to extend the MySQL term tables that the MathWebSearch *Crawler updates, in order to allow for the storage of information corresponding to an eventual idiom that the math term would be part of. Thus, I adapted the* MathWebSearch *Crawler for communication with the Idiom Spotter and modified the interface of the latter in order to output results in a predefined format.*

After the extraction of each XML math node from the currently analyzed document, the Crawler makes a call to the Idiom Spotter, providing the complete document and the XPath of the extracted math node. The Idiom Spotter then analyzes the math node and, if it finds that it is part of an idiom, updates the idiom database and provides as output a pointer (represented by an idiom id) back to the Crawler. Then, in the indexing phase, the Crawler adds this extra bit of information to the extra fields of the math term table in the MySQL database, in order to keep the connection between the term and the idiom in the idiom database. In case the term is not part of an idiom, then the output of the Idiom Spotter is blank and so are the corresponding table fields in the database, signaling that the math term does not point to any idiom.

5.2.4 Database Servers

As mentioned in the initial description, the database servers are represented by a MySQL server, with specific databases for each Search Node. The database corresponding to each Node holds a table of *documents* and a table of *terms*. The first table holds information regarding all the indexed documents while the second holds all the indexed math terms, with identifiers connecting each of them to the document from the first table that they belong to. The document identifiers are also stored in the Search Server substitution-tree index, as explained earlier. The database servers also hold a *sessions* database, which stores information about the active searches to the system. This enables the existence of a limited number of concurrent searches with caching of search results for improved efficiency and less strain on the system. The *sessions* database also holds information about each individual search result from all open sessions. This allows for easier and faster result handling at search time, which is very important for an online user system.

At *indexing* time, a document that contains at least one math term is added to the *documents* table and given an identifier. All of the math terms found inside it will also be added to the *terms* table as both XML and string representations with the document identifier that they are part of and their ordinal location inside it. The string terms are then all passed down to the Search Server for the second stage of *indexing*. At *search* time, the Web Server interacts directly to the Database Server, storing new session information at every search and the current search results identifiers for all open sessions in the *sessions* database. The Search Node databases are also queried for the mathematical terms and their documents of origin to display to the user on the results page.

Design Decision and Contribution *As mentioned above, the need to modify the existing* MathWebSearch *databases in order to add* Applicable Theorem Search *functionality and at the same time keep the* MathWebSearch *usability led me to make an extension of some of the* MathWebSearch *tables. Since the* MathWebSearch *Web Server only extracts the desired fields from a specific Search Node database, I chose to extend the "terms" table in each of them to contain more fields, storing information that links the term to an idiom in the idiom databases. These fields remain empty for terms which are not linked to any idiom. I also modified the "results" table in the* **sessions** *database in a similar manner. Thus, the functionality of* MathWebSearch *was not hindered, as the Web Server can retrieve search results without caring if the terms belong to an idiom or not, making possible the coexistence of* MathWebSearch *and* Applicable Theorem Search *on the same database server. This was an important design decision which avoids the redundancy of having the mathematical term information replicated in the idiom databases.*

5.2.5 Web Server

The application that provides an API and direct communication to the user is the Web Server. The main role of the Web Server is to await incoming requests from the user and to provide outgoing results back. Internally, the Web Server communicates to the Meta Server and the Database Servers. More pragmatically, the Web Server receives an incoming query from the user with a math term in XML format. Then, it transforms this term into string representation and sends the query to the Meta Server to search in the math index. It receives a set of results containing identifiers that point to the XML terms and the document links that are to be retrieved from the Database Servers. The Web Server then queries the Database Servers for these results, and outputs them in a certain format to the output. Each individual result contains a link to the document of origin, some metadata of the document (title, author, description, etc.) and the XML term matched. The Web Server API is available on a socket and predefined port[1]. The Web Server can be considered the core of the MathWebSearch system, being the central connection point between the GUI (Web Interface), the math index (Search Servers) and the document and term index (Database Servers).

As explained in the previous paragraph, the Web Server is only part of the *searching* stage of the search engine functioning. It needs to be constantly running in order to accept connections from the client side. These connections can be query connections, the workflow of which was explained above. The connections can also be targeted at retrieving only the details for a single result hit, in which case the Web Server doesn't need communicate to the Meta Server, all the details being stored in the Database Servers. The workflow of the Web Server is designed in such a way to communicate with the Search Nodes only for new queries, in order to increase the efficiency of the search process.

Design Decision and Contribution *Since I modified the Database Servers in order to store extra information identifying the possible idiom that a math term is part of, I also needed to extend the Web Server to retrieve this information for the user. I maintained the input side of the program the same, a query consisting of a math term, just like in a classic* MathWebSearch *search. This term is transformed into string representation and sent to the Meta Server, which sends back the locations of the XML terms in the Database Servers. As explained in Section 5.2.4, the term databases now contain pointers to the idiom databases for each term that is part of an idiom. Thus, the Web Server narrows down the results received from the Meta Server (and which would be fully outputted in the case of a* MathWebSearch *search) by only extracting those terms which have an idiom pointer from the the Database Servers. The Web Server then extracts the idiom and its component parts from the idiom database for each match result, following this pointer. The set of results is then outputted just like before, this time with the extra idiom information for each one. Thus, the basic functionality of the* Applicable

[1]At the moment, the MathWebSearch API is available at `raspberry.eecs.jacobs-university.de:19843`

Theorem Search *system has been acheived, with the premise that the Idiom Spotter and the idiom databases are reachable.*

5.2.6 Web Interface

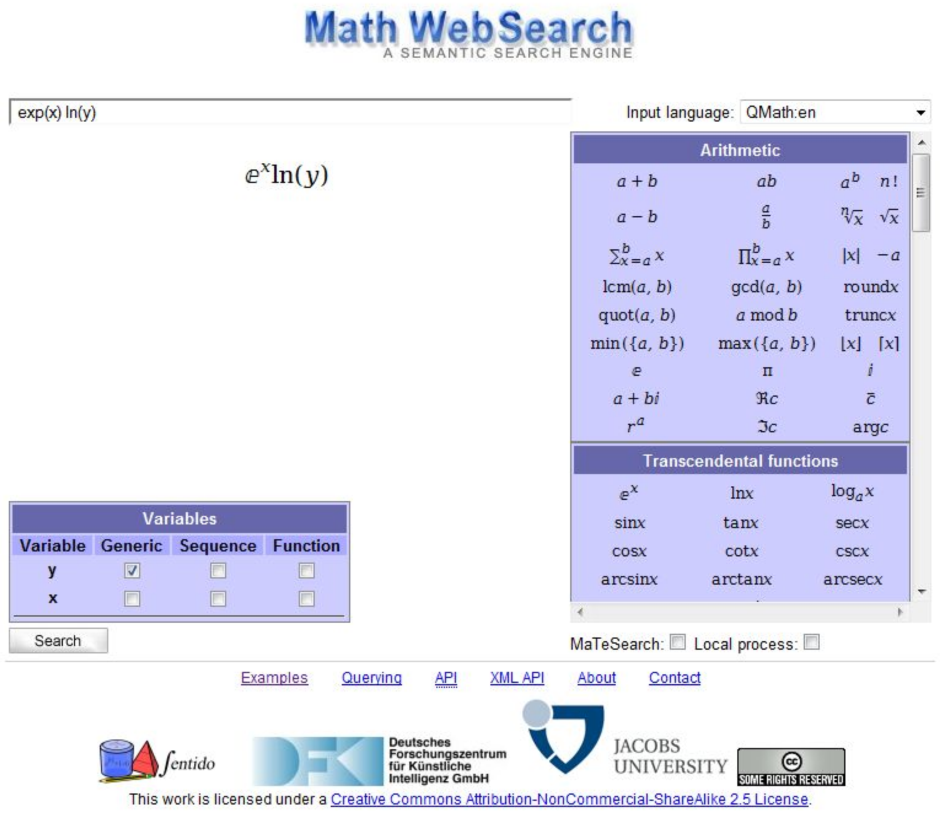
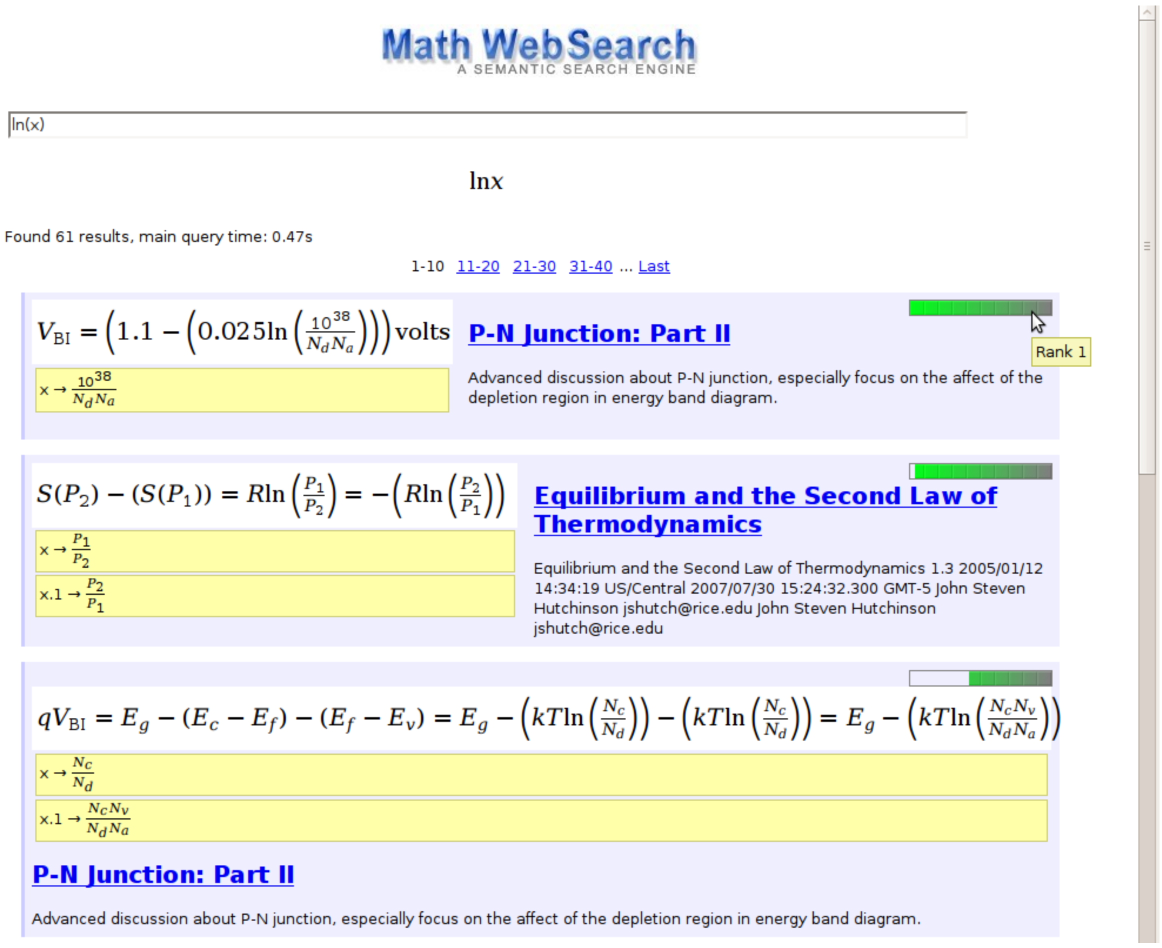

Math WebSearch
A SEMANTIC SEARCH ENGINE
exp(x) ln(y)
$e^x \ln(y)$
Input language: QMath:en
Arithmetic
Transcendental functions
Variables
Variable Generic Sequence Function
y
x
Search
MaTeSearch:
Local process:
Examples Querying API XML API About Contact
sentido
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
JACOBS UNIVERSITY
SOME RIGHTS RESERVED
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License.

Figure 5.3: MathWebSearch page at `search.mathweb.org`

The graphical user interface of the *MathWebSearch* system [ATS] is very important considering this is a system which takes mathematic formulas as queries. Taking advantage of the features of the *Sentido* formula editor [Pal06] (see Figure 5.3), currently available on the page, the user is be able to use a visual editor to enter the math formula to search for, but he/she can also enter Content MathML directly and use the *MathQuery* [KS06] query language to wrap it up in a valid query.

The Web Interface is the only user-driven and user visible part of the *searching* stage. It sends the user query to the Web Server and waits for the results. Once it gets the number of results available, it queries for details for each of them so as to display them in a fashion of 10 per page. Each hit contains the title of the document where the match was made, a link to the web page of the document and a short description of the contents. As you can see in Figure 5.4, each result also contains the matched term at the top left plus the substitutions which were applied on the user query (at the top of the page) in order to reach this term. Clicking on the title will load the page of the respective document.


Math WebSearch
A SEMANTIC SEARCH ENGINE

ln(x)

$\ln x$

Found 61 results, main query time: 0.47s

1-10 11-20 21-30 31-40 ... Last

$V_{\mathrm{BI}} = \left(1.1 - \left(0.025\ln\left(\frac{10^{38}}{N_dN_a}\right)\right)\right)$ volts

$x \to \frac{10^{38}}{N_dN_a}$

P-N Junction: Part II

Rank 1

Advanced discussion about P-N junction, especially focus on the affect of the depletion region in energy band diagram.

$S(P_2) - (S(P_1)) = R\ln\left(\frac{P_1}{P_2}\right) = -\left(R\ln\left(\frac{P_2}{P_1}\right)\right)$

$x \to \frac{P_1}{P_2}$

$x.1 \to \frac{P_2}{P_1}$

Equilibrium and the Second Law of Thermodynamics

Equilibrium and the Second Law of Thermodynamics 1.3 2005/01/12 14:34:19 US/Central 2007/07/30 15:24:32.300 GMT-5 John Steven Hutchinson jshutch@rice.edu John Steven Hutchinson jshutch@rice.edu

$qV_{\mathrm{BI}} = E_g - (E_c - E_f) - (E_f - E_v) = E_g - \left(kT\ln\left(\frac{N_c}{N_d}\right)\right) - \left(kT\ln\left(\frac{N_c}{N_d}\right)\right) = E_g - \left(kT\ln\left(\frac{N_cN_v}{N_dN_a}\right)\right)$

$x \to \frac{N_c}{N_d}$

$x.1 \to \frac{N_cN_v}{N_dN_a}$

P-N Junction: Part II

Advanced discussion about P-N junction, especially focus on the affect of the depletion region in energy band diagram.

Figure 5.4: MATHWEBSEARCH result page at `search.mathweb.org`

Design Decision and Contribution *Since the* APPLICABLE THEOREM SEARCH *system is identical to the* MATHWEBSEARCH *system at query time, the search page does not need to be changed. I adapted the results page, though, to also show the idiom result and the idiom component parts for each hit. This includes a change to the Web Interface itself, in order to handle the new output of the Web Server, and to display the extra information.*

5.2.7 Variable Spotter

As MATHWEBSEARCH evolved to version 0.4, new search capabilities like generalization, variation and unification query capabilities were added to the system. [KAJ+08] (see Section 3.6). The Variable Spotter, created by the author of this thesis at the time of MathWebSearch 0.3, is an addition to the system, made in order to help identify *universal variables* in the input documents. The program itself is called by the MathWebSearch Crawler at the beginning of the indexing stage. It then goes through the input MathML formulas and annotates all the variables with unique identifiers, before these terms are stored in the index.

Universal variables are mathematical terms bound by universal quantifiers, terms which can be unified with a search query term (as defined in Section 3.4). Universal variables are important for the *indexing* stage. As mentioned before, MathWebSearch keeps a substitution-tree index where every node contains a term. This is useful for instantiation queries, where we try to instantiate universals in the query with specific terms. However, in order to do generalization or unification queries, we need to look through indexed terms that are stored as universals and then search for a substitution that will unify them with the query terms. The Variable Spotter's task is to identify the status of variables in the input documents, so that they can be added to the math index as either universals, (existentially) bound variables or constants, allowing unification, generalization and variation queries at search time. The annotation is done as part of the *indexing* stage of the system, before the documents are searched for mathematical terms. The Variable Spotter does not discard context variable information. For example, $f(x) + 2 = x * y$ will be transformed into `@10(@11) + 2 = @11 * @13`, before being turned into internal string prefix representation. The addition of universal terms to the index is of particular importance for the Applicable Theorem Search system, which is based on generalization search.

The annotation *per se* is done with the help of a MathQuery [KS06] attribute. This attribute is added to all subterm variables in the math formulas which are part of the analyzed document. In the case of Content MathML, for example, this means that every `<m:ci>` node is spotted and enriched with an attribute with a unique identifier value.

The current implementation of the Variable Spotter determines the bound status of variables by only looking at the mathematical formula scope in which they are found. As mentioned in Section 3.4, this analysis is incomplete and can often create false positives, when the binders are outside of the mathematical formula scope. Such cases can be solved by looking at the whole context (scope) of a particular term. Since this context is often difficult to establish and it is contained in informal mathematical and language discourse, *context-based* disambiguation methods must be employed, in order to first identify all occurences of the same variable and then to try to extract the semantics of the variable from one of them. This involves a thorough linguistic analysis and is considered for future work.

Design Decision and Contribution *The Variable Spotter plays an important role in the envisioned* Applicable Theorem Search *system, as it enables generalization search, the basis for applicable formula search. Future improvements of the Variable Spotter in cooperation with the* Idiom Spotter *aim at extracting semantics from combined linguistic and* *mathematical context, through concepts like context idioms, presented in Section 3.4.*

Having described all the architectural components of the MathWebSearch system, we have gathered a collection of design decisions, taken to extend it towards the creation of the Applicable Theorem Search system. We will now give a short account of the performance of the MathWebSearch system, at version 0.4 and then look at how to approach the extension problem given experience gathered so far.

5.3 Performance

The MathWebSearch system is currently running on the Connexions [CNX09]and Wolfram [Wol09] repositories. Together, these two span over 93,000 documents and yield a math index of 1.5 milion subterms (memory footprint 400MB combined). Typical query execution times are in the range of miliseconds. Experimental searches have shown that the size of the query does not influence the search time.

It is expected that the extension of MathWebSearch to the Applicable Theorem Search system will impact search-time performance. The time needed to query the databases is naturally longer as extra calls need to be made in order to extract idiom details from the idiom databases, aside from the matched math term details.

The capabilities of the MathWebSearch system are very useful in the world of mathematical knowledge. However, they can not be fully employed without a large database of mathematical terms. The Connexions platform is the only database which provides user-generated Content MathML. However, not all of this content is formally correct. This is due to human error but also due to the fact that users have to make quite an effort to provide the correct content representation of their mathematical terms. The Wolfram database of mathematical functions is extensive but rarely offers more than just mathematical terms, without any textual context.

The largest problem of the MathWebSearch at the time of this thesis is the lack of proper semantic mathematical input. Databases like the Cornell arχiv of scientific publications or the converted arXMLiv version of it seem perfectly fit as input for the math search tool. However, the representation of the mathematics within is still only in LaTeX or Presentation MathML form, with little or no semantics. Projects like the LaMaPUn architecture, described in Sections 2.1.4 and 7.3 provide hope for future transformations to correct mathematical semantics of these documents [GJA$^+$09], thus empowering systems like MathWebSearch and Applicable Theorem Search.

5.4 Extensions

The creation of the APPLICABLE THEOREM SEARCH system has been previewed, piece by piece, in Section 5.2, as an extension built on top of the core functionality provided by MATHWEBSEARCH. A previous extension to MATHWEBSEARCH has been made in the past by the author of this thesis. Let us now take a look at it, in order to gain some knowledge about good practices of such extensions. Then, we will analyze the similarities between the past extension and the proposed extension to APPLICABLE THEOREM SEARCH.

The MATHWEBSEARCH system is a ongoing research project that has a lot of possibilities for extension and improvement. Since its creation, in 2006, it has enjoyed the attention of several researchers, all concerned with improving different parts of the architecture presented in Section 5.2. The first MATHWEBSEARCH extension was the addition of combined mathematics and text search capabilities, introduced by the author of this thesis in [Anc07].

5.4.1 MaTeSearch

The math search capabilities of the system become more powerful by allowing a query of a string and a formula at the same time, thus leading to more precise results. The system that provides the joint search capabilities has been called MATESEARCH. The name comes from a shortened version of “Mathematics and Text Search” and is pronounced [/’mate/] search in IPA notation.

For example, let us consider the use case where an engineer who has graduated a few years back from college, needs the formula for the probability density function (PDF) of two random variables $Y = X_1 + X_2$ on his new project (example described in [KAJ$^+$08]). The formula that the engineer is looking for is actually $f(y) = \int f(y|x_1)f_1(x_1)$, using marginal probabilities. But the engineer only remembers something about needing the joint PDF of the sum and one of the variables to calculate $f(y)$. Since the engineer doesn’t remember the exact formula for the joint PDF $f(y, x_1) = f(y|x_1)f_1(x_1)$ either, he would like to enter the search query $@f(@x, @y)$, which would match a large part of the formulas in the index and is therefore unsuited for searching. With the text search functionality, the engineer adds the string query `"random variable"`, to help narrow down the search. Of course, there are many documents which contain the word *variable* in them, but only few that will also contain the specified formula. The returned intersection results all fit in one page, with the document entitled *”Sums of Random Variables”* listed near the top of the first page of results. We see that even if both the formula and string queries are very vague, the intersection result set is narrowed down to a handful of documents, which can be quickly browsed over in order for their relevance to be determined for the user.

The combined math and text search facility is realized in by combining MATHWEB-

Search with the Nutch system, a text-based search engine built on the open-source Lucene [The06] architecture. MaTeSearch keeps an additional Lucene-based text index of the same corpus indexed by MathWebSearch. A MaTeSearch search performs a math formula query using the Web Server API (as described in Section 5.2.5) and text search query using the JAVA Nutch API in order to obtain two different sets of results which are merged by intersection and then presented as output. A combined "math+text" search works as follows [KAJ+08]:

- Use the math query on MathWebSearch and get a ranked set of results R^M
- Use the text query on Nutch and get a ranked set of results R^N
- Intersect the two result sets $R^M \cap R^N$ by ranking heuristic and supply the result list

The question of ranking search results for formula queries is largely unexplored territory, especially in the context of combined mathematics and text search. [You06] Currently, several approaches for formula search results ranking are being explored: match frequency, substitution size, familiarity of substituted constants, formula-class (prefer definition / theorem / . . .), formula-rank (prefer formulas that are frequently re-used/referenced). The ranking heuristic needs to look at the relevance of the results by analyzing the result sets from the two individual searches with respect to size and distribution. In the simulations attempts so far, Gaussian-modelled distributions have been used, but the ranking topic is still open for research.

5.4.2 Searching for Applicable Theorems

The MathWebSearch system was chosen as the backbone for the Applicable Theorem Search system because of its extendability and its ability to perform mathematical content search. Since the idiom patterns of interest are those containing mathematical formulas, this functionality was found "free of charge" in MathWebSearch. Also, the MathWebSearch system offers an already working search engine, with complete functionality from crawling to searching. The extensions needed to integrate it with the Idiom Spotter consist of extending this functionality to achieve a working combined platform.

As mentioned in the previous section, I have already had experience with adding extensions to MathWebSearch. If for the MaTeSearch system a separate text search engine was added, this time the Idiom Spotter needs to be run in parallel to MathWebSearch at *indexing* time and create a database of idioms. The difference in design however, is that the connection between the idioms and the mathematical terms has to be kept at the scope level of the sentence where the term is located, in comparison to the connection at file level between the string and math results. In pragmatic terms, the MathWebSearch and Nutch systems perform *crawling* and *indexing* separately and are only connected for *searching* by MaTeSearch. The Applicable Theorem Search system is built up as an integration of its two components, with functionality extensions made to all three stages, as shown in Section 5.2. Implicitly, the extension of MathWebSearch in the case of Applicable Theorem Search was much more invasive, needing a through analysis of the code and re-engineering of the base functionality.

Now that we have gathered all the information required for the understanding of the Applicable Theorem Search system and analyzed its two components, we will look at the practical application in the next chapter.

Chapter 6

Applicable Theorem Search

In the previous two chapters, I have presented two tools which perform natural language and mathematical content processing. This chapter presents the Applicable Theorem Search system, a combination of Idiom Spotter and MathWebSearch into a system which provides applicable theorem retrieval services.

The Applicable Theorem Search system is written in C++ and Perl, with a web GUI using PHP, Javascript and XSLT. Like MathWebSearch, it is released under the Gnu General Public License [Fre91] and its source code can be found in the *mathweb.org* repository, at `https://svn.mathweb.org/repos/mws/branches/Stefan`. In this chapter, I will use the term **idiom** with the restricted meaning of **theorem idiom**, the only idiom category harvested by the system.

In the following sections, I will provide a general description of the system, present the design decisions taken and the general process of search in the system. I will also show the MathWebSearch-like user interface, discuss the experimental results and conclude with the current problems encountered by the system.

6.1 Description

Through *generalization search* of mathematical queries with constants, the Applicable Theorem Search utility can retrieve a general formula in a theorem from an instantiated term (see Section 3.6 for more details). The typical use case for such a system is a scientist entering a mathematical formula with constants into the system and obtaining a list of results showing applicable theorems to this formula (see the running example in Section 1.2).

The Applicable Theorem Search system empowers the Idiom Spotter utility described in Chapter 4 with retrieval facilities, by linking the index of mathematical terms with that of natural language idioms. The system provides a proof of concept for the usefulness of theorem search in scientific documents, with certain limitations introduced by its two components. The realization of the Applicable Theorem Search system also proves the extensibility of the MathWebSearch system and the design flexibility of performing different types of searches on the same database.

6.2 System Design

The Applicable Theorem Search system is designed as an extension of the MathWebSearch system, with a connection point to the Idiom Spotter *at indexing time* and to the idiom databases *at search time*. The code extension was designed in such a way that it would intrude as little as possible into the original MathWebSearch implementation, for the purpose of keeping the math search functionality intact. The end result is a system which can run at the same time as MathWebSearch, using the same index and core math search server.

Motivated by the need to provide an extension which would change the math search engine into an theorem idiom search engine, Section 5.2 offers a thorough description of the MathWebSearch architecture, its components and what needed to be changed to reach the functionality of the Applicable Theorem Search system. Gathering all the **Design Decision and Contribution** paragraphs in the afore-mentioned section, we can put together a detailed description of the architectural components of the Applicable Theorem Search system. Practically, the Applicable Theorem Search system has conceptually the same architecture as MathWebSearch, with the exception that for every *Search Node* (Section 5.2.1), there is a corresponding idiom database and that the *Crawler* (Section 5.2.3) connects to the Idiom Spotter at indexing time. Referring to the architectural components discussed in Section 5.2, the Applicable Theorem Search system depends on the idiom databases, the XML term databases, the math search server, the crawler and the web server.

In the next section, I will try to illustrate the mode of operation of the Applicable Theorem Search system, by showing the lifecycle of a theorem idiom through the search engine.

6.3 A Run Through the System

We will now give a practical overview of the Applicable Theorem Search functioning process, by showing the running example in Figure 1.2 pass through all stages of the system. The three stage search engine approach described in Chapter 5 is only extended at the *indexing* and *searching* stages, as explained in Section 5.2. We will now go through these stages separately, showing how the Applicable Theorem Search works on the running example, which is given a fictitious location.

Indexing

1. After the *crawl* is complete, a **document** is selected by the modified version of the *Crawler* described in Section 5.2.3 and its XML is parsed into a DOM **tree**.

   The document currently analyzed contains the theory about Master Theorem, a paragraph of which is:

   Case 1: If given a recurrence relation of the form

   $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

   with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then

   $$T(n) = \Theta\left(n^{\log_b a}\right)$$

2. The **tree** is searched for all nodes containing mathematical terms and the **node** with the following **term** is found:

   $$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

3. The **document** information is stored in the *documents* table of the MathWebSearch databases:

   - **Title:** *Master Theorem*
   - **Location:** `www.someurl.com`

4. The **document** and the **XPath** to the math node found above is sent to the Idiom Spotter.

5. The Idiom Spotter analyzes the **scope** of the **node** and extracts the entire **sentence**:

   If given a recurrence relation of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$

6. The sentence is matched against the *if H then C* **idiom pattern**

7. The **idiom**, its components and **location** information are stored in the *idiom database.* The extracted components are:

   - **Idiom:** *if H then C*

- **Hypothesis:** a recurrence relation of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$
- **Conclusion:** $T(n) = \Theta\left(n^{\log_b a}\right)$
- **Location:** `www.someurl.com`

8. The Idiom Spotter returns an **idiom identifier** (or **pointer**) back to the *Crawler*.
9. The *Crawler* transforms the **term** into string representation and stores the **XML term**, **string term**, **idiom identifier** and **location** information (not relevant here) into the *terms* table of the MathWebSearch databases.

[9'.] In case the Idiom Spotter does not find an idiom that the math **term** is part of, the **idiom identifier** field(s) is left blank.

10. [Steps 3 - 9 are repeated for all mathematical terms in all documents crawled.]

11. After all the terms from all documents have been analyzed, all the **math terms** in string representation are fed into the *Search Server* (see Section 5.2.1) to create the *substitution-tree index*.

Searching

1. The user accesses the Applicable Theorem Search *Web Interface* at `http://betasearch.mathweb.org` and enters the mathematical **query**:

$$C(t) = 9C\left(\frac{t}{3}\right) + r(t)$$

2. The *Web Interface* sends the **query** as a **generalization search** to the Applicable Theorem Search *Web Server*
3. The *Web Server* converts the generalization **query** into string representation (not relevant here) and sends it further to the *Meta Server*
4. The *Meta Server* sends the string math **query** forward to the *Search Servers* that it manages.
5. The *Meta Server* gathers the individual **results** from them and sends the aggregated list of **matched terms** back to the *Web Server*
6. The *Web Server* filters the **matched terms** and extracts only those with an **idiom pointer**

7. For each **matched term**:
   (a) Extract **term** details from MathWebSearch databases.
   (b) Extract **idiom** details from Idiom Spotter databases.
   (c) Send **detailed results** to *Web Interface*
8. The *Web Interface* displays the **detailed results** from the *Web Server* to the user.

[8'.] One of the results displayed is:

- **Query Term:** $C(t) = 9C\left(\frac{t}{3}\right) + r(t)$
- **Document:** *Master Theorem* at `www.someurl.com`
- **Matched Term:** $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- **Substitutions:** $T \to C$, $a \to 9$, $n \to t$, $b \to 3$, $f \to r$,
- **Idiom**: *if H then C*
- **Hypothesis**: *given a recurrence relation of the form* $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, *with* $a \geq 1$, $b > 1$ *and* $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ *for some constant* $\epsilon > 0$
- **Conclusion**: $T(n) = \Theta\left(n^{\log_b a}\right)$
- **Scope**: If given a recurrence relation of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

With the search results listed above presented in the Web Interface shown in Figure 6.4(which depicts a simpler search example), all the required information to solve the problem of the scientist introduced in Section 1.2 has been retrieved. Going back to the running example situation, the biologist will first look at the **matched term**, showing the mathematical formula that was instantiated to her query term. By plugging in the **substitutions** mentioned below, she can verify that her **query** (also shown at the top of the page) actually fits this generalization. Once she is convinced that the search result is useful, she can check the returned theorem, named *"Master Theorem"* and explicitly presented as an *if H then C* **idiom**. The individual parts (**hypothesis** and **conclusion**) are given separately but the whole containing **scope** is also included in the search results. Note that for this particular theorem result, she has to verify the extra conditions $a \geq 1$, $b > 1$, which hold true for $a = 9$ and $b = 3$. The **hypothesis** also contains a condition on the structure of $f(n)$, so one more thing that she needs to check is that $r(t) = \mathcal{O}\left(n^{\log_3(9)-\epsilon}\right)$. From her previous research, she knows that $r(t)$ is linear, so $r(t) = \mathcal{O}(n)$ for $\epsilon = 1 > 0$. Now that she seen that her query term actually fits all the conditions mentioned in the hypothesis, she takes the **conclusion** and applies it to her problem instance. By plugging in the values for a and b, she effortlessly finds the solution to her problem, the upper bound for the new cell decay rate, namely $C(t) = \Theta(n^2)$. Having not heard of this theorem before, the biologist is curious (and maybe a little skeptical) so she clicks on the given link, which loads the online document where the idiom was found, displaying the paragraph where the theorem is given and proved.

Thus, we have given a complete run of the Applicable Theorem Search system, and have shown how it provides a semantic solution the running example problem defined in Section 1.2.

Note that the algorithms steps shown above for the *indexing* part of the system represent an non-intrusive extension to the MathWebSearch system. Thus, both the substitution-tree index and the term databases that are created by this algorithm are directly usable by the MathWebSearch system at *search* time, without any code modification.

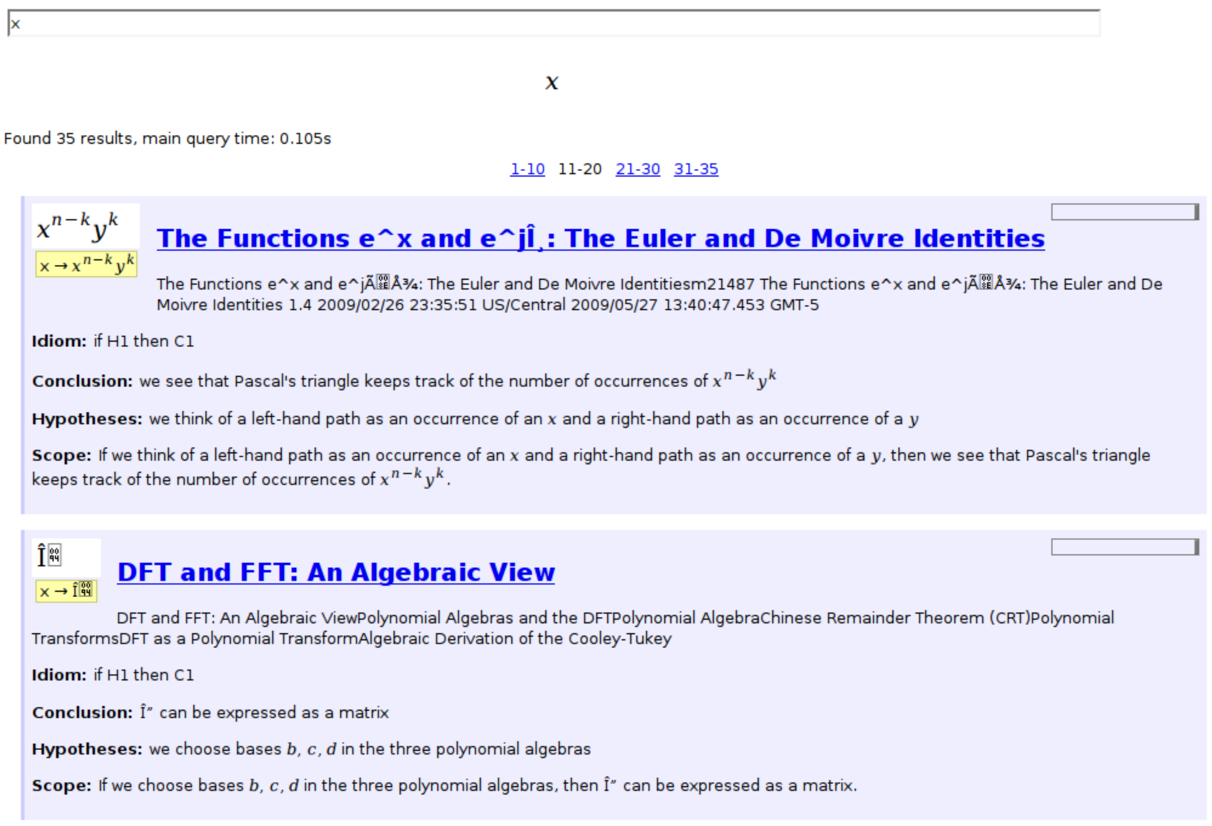
The graphical user interface for the search algorithm described above is a web page, very similar to the one used for the MathWebSearch system.

6.4 Web Interface

A clear presentation of the search results retrieved from the databases by the search algorithm listed above is very important for the usability of the system. The prototype user interface of the Applicable Theorem Search system is borrowed from the existing *MathWebSearch* system query page [ATS] and tweaked to fit the requirements of the current project. It is currently available at `http://betasearch.mathweb.org`. As mentioned in Section 5.2.6, the search page looks exactly the same as the one used by MathWebSearch (shown in Figure 5.3), since the users only query for mathematical terms. Thus, the query language is the same.

The results page is slightly different than the one shown in Figure 5.4. As shown here in Figure 6.4, the results presentation is similar, but extended to bring information about the theorem idioms related to the matched term. The figure shows a simple illustrative example of a search on a restricted test index of the Connexions [CNX09] corpus. Presenting both the semantics of the math substitution and of the natural language pattern, each result shows the mathematical terms and the substitutions on the left and the idiom details in the center, below the title and description of the document. For example, the first result showed in Figure 6.4, shows the matched idiom pattern *if H1 then C1*, with the conclusion, hypothesis and scope mentioned below. It is always useful to show the scope, as it presents the whole sentence which was matched against the idiom pattern shown at the top of the result item.

A simple clickthrough will allow the user to reach the presentation form of the document of origin for the idiom and an eventual XPointer or XPath (where available) will point the browser directly to the paragraph where the idiom is located.



x

x

Found 35 results, main query time: 0.105s

1-10 11-20 21-30 31-35

$x^{n-k}y^k$ $x \to x^{n-k}y^k$ **The Functions e^x and e^jÎ¸: The Euler and De Moivre Identities**

The Functions e^x and e^jÃÂ¾: The Euler and De Moivre Identitiesm21487 The Functions e^x and e^jÃÂ¾: The Euler and De Moivre Identities 1.4 2009/02/26 23:35:51 US/Central 2009/05/27 13:40:47.453 GMT-5

Idiom: if H1 then C1

Conclusion: we see that Pascal's triangle keeps track of the number of occurrences of $x^{n-k}y^k$

Hypotheses: we think of a left-hand path as an occurrence of an x and a right-hand path as an occurrence of a y

Scope: If we think of a left-hand path as an occurrence of an x and a right-hand path as an occurrence of a y, then we see that Pascal's triangle keeps track of the number of occurrences of $x^{n-k}y^k$.

Î $x \to$ Î **DFT and FFT: An Algebraic View**

DFT and FFT: An Algebraic ViewPolynomial Algebras and the DFTPolynomial AlgebraChinese Remainder Theorem (CRT)Polynomial TransformsDFT as a Polynomial TransformAlgebraic Derivation of the Cooley-Tukey

Idiom: if H1 then C1

Conclusion: Î” can be expressed as a matrix

Hypotheses: we choose bases b, c, d in the three polynomial algebras

Scope: If we choose bases b, c, d in the three polynomial algebras, then Î” can be expressed as a matrix.

Figure 6.1: Applicable Theorem Search results page at `betasearch.mathweb.org`

Having showed the main algorithms behind indexing and searching and how the system interacts with the user, let us now evaluate the performance of the system as a whole, in the following section.

6.5 Performance and Test Corpora

The only test case for the evaluation of the Applicable Theorem Search system so far was the Connexions repository [CNX09], which is also indexed by MathWebSearch. The three arXMLiv-based corpora mentioned in Section 4.4.1 were not indexed, because of the high probability that the Content MathML representation of the math terms found within is semantically flawed (see Sections 2.1, 7.3). The Wolfram functions corpus [Wol09] (also indexed by MathWebSearch) is also unfit as input, as it contains mostly only formulae and no text (thus no idioms). The theorem idiom patterns which have been used for idiom spotting are presented in Table 6.1, with the same information about the number of occurences presented in Table 4.2.

We also present some interesting statistics about the indexed corpus in Table 6.2. There,

Idiom pattern	**Frequency Connexions**
assume H1 then C1	29
given H1 then C1	43
H1 if and only if C1	56
H1 implies C1	170
C1 only if H1	102
C1 only when H1	35
if H1 and if H2 then C1	24
if H1 and if H2 then C1 and C2	10
if H1 then C1	**1161**
let H1 then C1	61
suppose H1 then C1	23

Table 6.1: Theorem Idiom Patterns used for Idiom Spotting on the Connexions Corpus

you can see the number of mathematical terms in the corpus, the number of theorem idioms (which naturally contain math terms) and the number of files that have either of the two. Comparing the number of terms found in idioms with the number of idioms is interesting, as it gives quite a high average number of mathematical formulae per idiom (12.9). This reinforces our assumption that theorem-like natural language patterns are heavily used to present facts containing mathematical formulae. One theorem idiom usually contains more than one mathematical term (usually, two or more). The low number of idioms found in the Connexions corpus is mostly due to the non-mathematical nature of the documents within (see discussion in Section 4.4.2) but it can partially be due to the restricted number of patterns used for this experiment and the limitations of the Heuristic Pattern Matching approach, as described in Chapter 4.

The performance of the Applicable Theorem Search system is not measured at *indexing* time. There, because of the external call to the Idiom Spotter for each individual mathematical term, the system is slower than the regular MathWebSearch. In fact, redundancy is introduced in the indexing process, since a document is loaded and parsed by the Idiom Spotter for each of its math terms. However, the indexing step does not need to be efficient, since it's an *off-line* periodic process. The Applicable Theorem Search system needs to be efficient and fast at *search* time. There, it performs slightly worse than the MathWebSearch system, responding to a user query within up to a second. This is expected (as the Web Server performs twice more MySQL queries as before, extracting both mathematical term details and idiom details from the databases), but considered acceptable for the purpose of this test case.

The general performance of the Applicable Theorem Search *system can not yet be gauged on the indexed corpus.* Idiom Spotter *experimental results presented in Section 4.4.2 clearly show that mathematical theorem idioms can be retrieved, even with the simple heuris-*

Statistics	**Connexions corpus**
Total files	11712
Files with mathematical terms	6241 (53%)
Files with th. idioms	431 (3.6%)
Files with th. idioms / Files with mathematical terms	6.9%
Total math terms found	86831
Total theorem idioms found	1714
Total math terms found in th. idioms	22158
Average math terms per th. idiom	12.9

Table 6.2: Applicable Theorem Search statistics on the Connexions corpus

tics and reduced list of idioms that the Idiom Spotter *employs at the time of this thesis. Experiments on the Saarbrücken corpus retrieved over 200000 theorem idioms while the Connexions corpus only yielded 1700. Clearly, the Connexions corpus is not the right database to test out the* Applicable Theorem Search *capabilities. However, it is the only database which can be added to the index at the time of this thesis because of the problems with mathematical formula representation in the* arXMLiv *corpus. This situation is unfortunate, since the* Applicable Theorem Search *is tied to the generalization search functionality of the* MathWebSearch *system, functionality that is only available on semantically correct content representations of mathematical terms. Thus, the* Applicable Theorem Search *system is a utility which can not be used to its full potential until we can provide it an idiom-rich input corpus, which can not be done yet at the time of this thesis. Research on semantically enriching presentation-oriented mathematical terms (Presentation MathML) towards correct semantics (Content MathML) is the most promising direction for the future of the* Applicable Theorem Search *system. A current research project directed at this development is the* LaMaPUn *architecture, presented in the following chapter, as an outlook for the future of the* Applicable Theorem Search *system.*

The next section will list some of the practical challenges encountered in the development of the Applicable Theorem Search system.

6.6 Challenges Encountered

Designing the Applicable Theorem Search system around MathWebSearch and the Idiom Spotter presented a lot of challenges, both in terms of the theoretical design decisions made to interconnect math and natural language, and the practicalities of connecting the code of two different systems written in two different programming languages. Some of these challenges have been overcome, others have remained, constituing pitfalls of the current theorem search approach. I will present some of these problems in this section, with the hope that they will be useful for the further development of the system or for other scientists who choose to approach the same topic.

Programming Languages and Code Comprehension The Applicable Theorem Search system was implemented in Perl, since the original MathWebSearch system was implemented in this programming language by Ioan Şucan and most of the original source files were just extended. My scripting language of choice is Python, in which I wrote the Idiom Spotter, which made the connection between the two components a little more difficult and less efficient. Although extending the existing code to reach the Applicable Theorem Search functionality was not a great challenge, getting familiar with the MathWebSearch implementation to the point of knowing where to make changes was a tiresome and time consuming task. Creating the prototype web interface for the Applicable Theorem Search system also required code comprehension, the original GUI being written by Alberto Gonzalez Palomo. The latter has created another layer of XML communication between the PHP code that connects to the web server and the results page which uses AJAX to display individual results.

Imported Library APIs Although both the Perl and Python scripts use the `libxml2` library for creating the DOM tree and performing operations on it, the two wrappers around the original C code have different APIs and even differ in functionality. For example, the Python implementation needs a `ParserContext` object before it can parse an XML string, while its Perl counterpart works just with a `Parser` class. Thus, seeing how the document tree is handled by MathWebSearch in the Perl code did not help much for the Idiom Spotter Python implementation.

Namespaces Since the extracted information is a mixture of XML math nodes and text (which most often runs across more than one node), we should be very careful when taking this data out of the original document. The mathematical nodes have the MathML namespace associated to them, but in XML text, this is most often declared just once at a top level node and then just used as a prefix throughout. When extracting the math node out of the document, we must remember to look up in the DOM tree all namespace declarations that it falls under and copy them into the math node or into a wrapper tag around the whole idiom.

XML vs String Processing Generally, the approach of using NLP methods on XML documents introduces a lot of problems related to content-unrelated tags like metadata tags, style tags, etc. While mathematical terms are represented as fully enclosed MathML nodes, the text composing an idiom is often spread out across several text nodes, even at different depths in the tree (because of tags related to presentation aspects). Analyzing documents in order to extract both at the same time raises the question of processing the information as DOM node objects (with class methods) or as a string (with regular expressions). The IDIOM SPOTTER uses both approaches, needing to do low level string analysis for pattern matching and DOM-level processing for namespace extraction, node comparison and mathematical node processing. Tools targeted specifically at XML documents are already being developed for future natural language and mathematics processing tasks, as part of the research in the LaMaPUn group [LaM09].

Chapter 7

Input Problem Outlook

In the previous three chapters, I have presented three systems which process scientific texts with mathematical formulae, all of them taking input in XML format. In evaluation discussions in all three chapters, it has become apparent that there exists a general input problem. The problem is not that we don't have input for the systems, but that the input comes in the wrong format. Most of the time, the interesting documents contain mathematics in Presentation MathML, instead of the desired Content MathML.

As the tests in Section 4.4.2 show, databases like the Saarbrücken corpus (or extrapolating, the ARXMLIV database) provide plenty of theorems and mathematical terms. However, these can not be processed and indexed as they do not come in a semantic representation. This is especially a big problem for the APPLICABLE THEOREM SEARCH system, which needs theorems with mathematics in a content format.

This chapter will describe the reasons that led to this input problem in the first place, describe the prerequisites for solutions and then project a practical solution for the future. The proposed solution is an ongoing research project which provides an architecture for purification and semantic enrichment of the mathematics in XML documents as an input pipeline for the APPLICABLE THEOREM SEARCH system.

7.1 Representation of Mathematical Terms

The input problem is the representation format for mathematical terms. Representing mathematics in scientific documents in the academic field is achieved most of the time with TEXor LATEX, Microsoft Office Equation Editor, or even images. LATEX is a high-quality typesetting system, including features designed for the production of technical and scientific documentation. It is the *de facto* standard for the communication and publication of scientific documents. [LaT09]. However, LATEX provides a presentation-oriented language, that strives for best appearance, but not for semantic rigurosity, since the end representation targets for LaTeX sources are PDF or printed documents, which are not processed further. Since most scientific articles are written in LaTeX (for ex., the arχiv articles are written only in this format), the input problem is created very early on. The transformation of scientific documents into XTHML webpages available on the Internet (see the LaTeXML project described in Section2.1.3) leads to the conversion of the mathematics from LaTeX formulae into MathML, an XML format compatible with Internet browsers.

MathML is a W3C recommendation for a low-level specification for describing mathematics as a basis for machine to machine communication on the Internet [W3C07]. MathML is an XML format which can be used to encode either the presentation of mathematical notation for high-quality visual display, or the mathematical content, for applications where the semantics is essential. The two subformats are called Presentation MathML, respectively Content MathML, with the obvious implicit purpose. Another standard for representing mathematical terms with their full semantics in an XML format is OpenMath [Ope09], which is more rigurous than Content MathML. However, most mathematics found on the internet are represented in Presentation MathML, which again creates the input problem. The reason why the mathematical formulae are written in Presentation MathML is probably because it is easier to create and it resembles LaTeX in its visually-oriented approach.

Both LaTeX and Presentation MathML add to the input problem. The following section will explain what makes Content MathML a useful representation and, as such, a solution to our input problem.

7.2 Useful Mathematical Representation

The service that MathWebSearch and Applicable Theorem Search provide is indexing mathematical formulae and providing search functionality. The completion of this service requires full understanding of the underlying mathematical structure of the indexed terms. Otherwise, subterms could not be extracted from the mathematical blocks, and the only search capabilities available would be string comparison based searches, which would bring no difference from usual string-based search engines like Google, and no semantics. In order to *understand* a mathematical term, the systems must parse and recognize its structure, just like parsing a text or XML document, transforming it from a string of characters into a logical object. This can only be done if the string of characters that represents this term follows a predefined grammar, which the mathematical parser recognizes. Thus, the only formats of mathematics that can be fully semantically understood by computers are those following predefined rules, like Content MathML or OpenMath. The MathWebSearch system and the Applicable Theorem Search system described in Chapters 5 and6, respectively, can only extract and index subterms from mathematics stored in these two formats.

But finding scientific papers or articles with *Content MathML* or *OpenMath* (a solution to our problem) is a difficult task, since most authors choose to write the simpler presentation(*Presentation MathML*) only or use the classical LaTeX to PDF path for exporting their documents. Both the *ArXiv* [arX07] and the *DLMF* [DLM05] collections of articles do not contain Content MathML by default. At the moment, one of the few fairly reliable sources of scientific content with *Content MathML* is the *Connexions* [CNX09] repository.

The MathWebSearch system, built in 2006 started off with an index of the Connexions repository as a test case of a corpus containing Content MathML. Since then, surprisingly, no other repositories with user-generated Content MathML have been added to the system, fact which slowly brough out the existence of input problem. Although the *W3C* has 15 MathML authoring systems listed on its recommendation page [W309], other large sources of user-generated mathematical content have not been found. The Wolfram repository, now also indexed by MathWebSearch, is a collection of mathematical functions transformed from *Mathematica* [Wol99] into Content MathML with the help of an automated converter. The indexing of this corpus directed the developers of MathWebSearch to turn their attention towards transformers, which could convert the mathematics in other corpora into the desired format.

Current research efforts are being focused on creating a clear, unambiguous translation from LaTeX math into Content MathML, with the LaTeXML system (see [Mil09], Section 2.1.3). The efforts are aimed at finding a solution to our input problem. The initial LaTeXML transformation of academic documents into browser-compatible entities targeted a good visual representation. Also, since LaTeX provides a language which is fully expressive only in presentational terms, the transition to Presentation MathML was natural. However, scientists today attempt to formalize and automatically retrieve as much knowledge as possible from these documents, written in a format inherently void of semantics (see Section 2.1.4). Thus, the need for a transformation to a semantic format is ever more pressing, and has sparked project initiatives like the one described in the following section. I will now present the LaMaPUn architecture, the envisioned future solution to the input problem for the Applicable Theorem Search system.

7.3 The LaMaPUn Architecture

The "Language and Mathematics Processing and Understanding" (LaMaPUn) project [GJA+09] is a recent endeavour of the KWARC research group at Jacobs University, led by the author and a few other graduate students [LaM09]. The project investigates semantic enrichment, structural semantics and ambiguity resolution in mathematical corpora, through an architecture built up around the output of LaTeXML. The most important long term goal of the architecture is to provide a tool for semi-automated mathematical content purification, correction and semantic enrichment. In simpler terms, the architecture would provide the means to turn the visually oriented, semantically ambiguous conversion result of LaTeX mathematical formulae currently outputted by LaTeXML [SK08] into semantically accurate Content MathML.

The LaMaPUn architecture provides an abstraction layer for distributed development of semantic analysis tools on the large arXMLiv corpus. This is achieved by a stand-off RDF abstraction of the analyzed documents, allowing the processing modules to plug in to the central blackboard and access structural information about the contents. The architecture, shown in Figure 7.1 encapsulates *preprocessing*, a "*Semantic Blackboard*" for distributed semantic analysis, a representation of the *semantic results*, appropriate *generation of output formats*, as well as *user interaction and visualization*, as first presented in [GJA+09].

7.3.1 Architecture Modules

The **Preprocessing** module tries to convert semantically inadequate mathematics, written in LaTeX for the pure purpose of presentation (and transported by LaTeXML into an XML format) into semantically correct terms. Its primary purpose is to separate the LaTeX normal text mode from the math mode, where the two are erroneously mixed. For example, it corrects text entries like "1^{st}", which put natural language inside math mode for the pure purpose of accessing visual operators (like "^"), creating a false math term. Also, entries like "{\bf x} - {\bf y}" (without $), which use presentational text methods to display math formula, which loses its semantics as a math formula because it is not in math mode. The preprocessor also tries to translate the natural language identifiers (found sometimes in mathematical mode as variable names) correctly into the XML representation. For example, if not corrected, the automatic LaTeXML translation would turn the presentation of $last \neq first$ into an inequality between a multiplication of four tokens and multiplication of five tokens.

In the heart of the architecture lies the **Semantic Blackboard** module, coordinating the analysis process and acting as an interface between the different semantic applications and the rest of the architecture. The Semantic Blackboard comprises a system based on a centralized RDF database which stores all transformed XML documents in the form of semantic annotations about their internal structure and contents. More specifically, all text and math content of a documents is stored in the database in the form of subject-predicate-object statements, accessible by special SPARQL queries. This abstraction layer over the documents allows **semantic analysis modules**(presented in the next section) to "plug in" to the Blackboard, extract the desired content, process it, and introduce the stand-off results back into the database.

The **Semantic Result** and the **Output Generation** modules make the transition between knowledge stored in the RDF database and the final output format. The semantic annotation results from the analysis modules stored in stand-off format in the Blackboard are aggregated and merged into the documents and the final transformation is made to reach the output XHTML or OMDoc [OMD] format. At this point, the semantic anno-

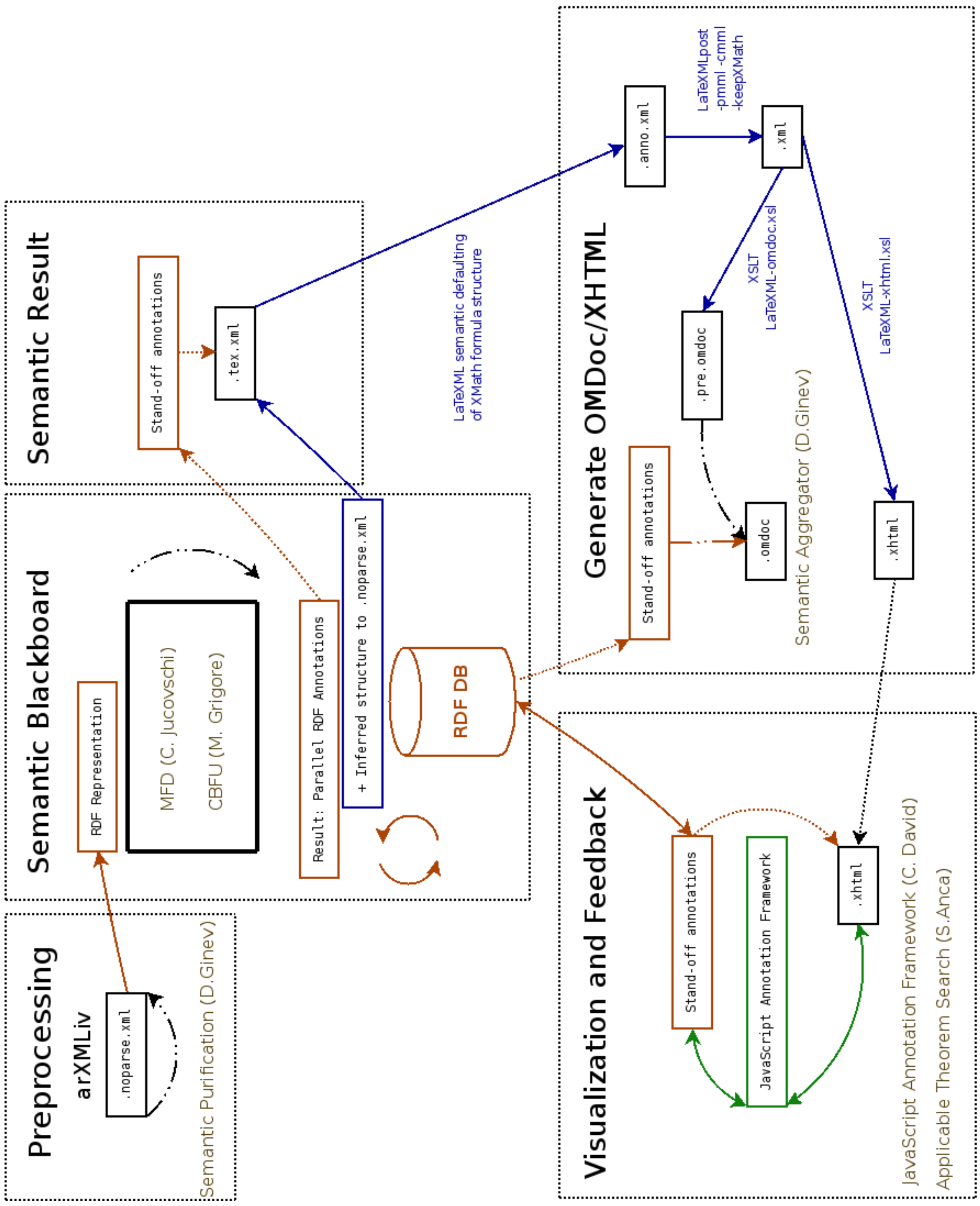

Preprocessing
arXMLiv
.noparse.xml
Semantic Purification (D.Ginev)
Semantic Blackboard
RDF Representation
MFD (C. Jucovschi)
CBFU (M. Grigore)
Result: Parallel RDF Annotations
+ Inferred structure to .noparse.xml
RDF DB
Semantic Result
Stand-off annotations
.tex.xml
LaTeXML semantic defaulting of XMath formula structure
Generate OMDoc/XHTML
.anno.xml
LaTeXMLpost -pmml -cmml -keepXMath
.xml
XSLT LaTeXML-omdoc.xsl
XSLT LaTeXML-xhtml.xsl
.pre.omdoc
Stand-off annotations
.omdoc
.xhtml
Semantic Aggregator (D.Ginev)
Visualization and Feedback
Stand-off annotations
JavaScript Annotation Framework
.xhtml
JavaScript Annotation Framework (C. David)
Applicable Theorem Search (S.Anca)

Figure 7.1: A high-end overview of an arXMLiv analysis architecture

tations gathered in the Blackboard provide enough information to turn the result of the LaTeXML transformation of LaTeX formulae into correct Content MathML, the desired target for our math and idiom search engines.

The architecture also incorporates the **Interaction and Visualization** module, a Javascript tool enabling user interaction and feedback. This module allows users to review the converted XHTML corpus articles and introduce annotations for mathematical formulae which they find to have wrong semantics. By involving human corrective input into the creation of semantics for the mathematic formulae, it can benefit the development of the LaTeXML software, by offering feedback with regard to possible conversion errors. Also, the existence of this communication channel will benefit the developers of semantic analysis modules by providing early feedback regarding the derived semantics.

7.3.2 Semantic Analysis Modules

The Semantic Blackboard described above provides a workbench for various analysis tools on the arXMLiv document. The semantic analysis tools that the LaMaPUn group is developing tackle the problems of mathematical formula understanding and disambiguation.

The **Context Based Formula Understanding** module tries to clear the context-based ambiguities that occur in mathematical texts, aiming to automatically retrieve information that can be deduced from the context, but that is intentionally omitted by mathematicians for the sake of succintness. For example, when the symbol ω occurs inside a mathematical text, it is necessary to first understand its meaning in order to understand the meaning of the symbol ω^{-1}. In the case when it is known to be a function, then ω^{-1} is obviously the inverse function corresponding to ω. This is completely different than the situation when ω is a scalar value and ω^{-1} should be understood as $1/\omega$. If the meaning of the symbol is apriori understood from the context, the semantics of a term like $\omega(x+2)$ later in the text is no longer ambiguous between a multiplication and a function application. The context based formula understanding module makes use of Word Sense Disambiguation techniques in order to deal with formula context. Using the RDF document representation provided by the Blackboard allows the decomposing of mathematical fragments to the symbol level. This allows the system to process the full underspecification of the formulas, which gives best results in detecting subformulae and in relating to context. The results of the disambiguation process are stored back as stand-off annotations and are fully accessible for subsequent use by other applications. For futher details, see [GJA^{+}09].

The **Mathematical Formula Disambiguation** module aims to disambiguate the parts of formulas which require little or no context information. For example, in $f(x) = x - 5$, f could be multiplied by x or function f is evaluated at x. For a computer processing this formula, both are possible, but a reader with any experience in reading mathematical texts would see the formula as unambiguous straight away, due to highly standardized symbol conventions. The disambiguation starting point is the manual creation of rules like: *if "(" is followed by "symbol" followed by ")", then "(symbol)" is an argument to a* *function.* Or *if "symbol1" is followed by "symbol2" in a subscript, then separate them by ","* as in $F_{ij} = F_{i,j}$. This method is very similar to the rule-based approaches employed to solve the Part of Speech Tagging (POST) problem [Bri95]. For now, these rules are created manually by observing certain common patterns in the documents and analyzing their effect after applying the rules. See [GJA+09] for more details.

7.3.3 Significance

The LaMaPUn architecture is a very promising project, providing support for semantic mathematical analyses but also any other type of computation on XML documents. For the Applicable Theorem Search system, the architecture is important because it proposes a solution to the input problem, by providing the mathematics in the correct format. With over half a million converted documents in 37 scientific subfields, the arXMLiv corpus is a tremendous and continuously expanding knowledge database. Being able to process all its documents would solve the input problem of the Applicable Theorem Search system. Indexing all the mathematical formulae and theorem idioms in this corpus would also provide an oustanding knowledge retrieval engine. Unfortunately, at the moment, the converted mathematics is mostly presentation-oriented and its content representation is flawed because of inherent ambiguities, making it unfit as input for the search systems. With the help of semantic enrichment modules and human input, the LaMaPUn architecture takes the step of transforming the presentation-oriented math into the correct unambiguous Content MathML representation, thus transforming the arXMLiv documents into correct input for the Applicable Theorem Search system. The LaMaPUn architecture can be seen as an input pipeline to the system, starting off from human created LaTeX sources and outputting XML documents with semantic mathematical content, thus providing input and solving the input problem.

Chapter 8

Evaluation and Future Work

This chapter will provide a combined evaluation of the IDIOM SPOTTER and the APPLICABLE THEOREM SEARCH systems and a general evaluation for the work performed in this thesis. The second section will list all the future work items for the two systems.

8.1 Evaluation

This thesis has set out to prove the claim that fixed-structure natural language patterns can be used to capture semantic knowledge from scientific texts with mathematical content. The proof of concept for this claim is given by the IDIOM SPOTTER system. The IDIOM SPOTTER is a system which extracts idioms from XML texts, based on the theory of idioms as semantic patterns. It provides a means to empirically check and refine the theoretical assumptions made in Chapter 3.

A second, more specific claim made in this thesis is that we can provide a utility that will help scientists find theorems which they can apply to a particular mathematical formula in order to make progress in their research. This would be achieved by storing theorem idioms in a database and retrieving them with generalization search on mathematical terms. The proof of concept for the second claim is given by the APPLICABLE THEOREM SEARCH system, a search engine using a combination of MATHWEBSEARCH as a math index and search engine and the IDIOM SPOTTER as an idiom retrieval tool.

The testing of the IDIOM SPOTTER has been performed by processing three different corpora of scientific documents, as explained in Section 4.4. Although still in its early phase of development, the implementation of the IDIOM SPOTTER described in Chapter 4 has extracted a high number of idioms even when testing with a restricted set of patterns. This gives us a positive initial intuition that general scientific texts contain such fixed-pattern formulations. The results obtained on the Saarbrücken corpus are very encouraging, proving that texts with pure mathematical content highly employ such structures. The high density of idioms in this corpus offers great opportunities for refining the heuristics. The percentage of files with idioms is average on corpora with uncertain mathematical content (42% of files in the WebDev sandbox), and low on corpora with little mathematical content (3.8% of files in the Connexions repository), as you can see in Table 4.1 . We assume that the lower percentages achieved so far for these corpora are due to lack of concentrated mathematical content but also to problems in the implementation of the heuristic idiom spotting method presented in Section 3.3.2 on XML texts. Improvements are suggested in Section 8.2. At the time of this thesis, the Idiom Spotter has an example idiom pattern list of about 15 simple idioms, mostly theorem patterns, shown in Table 4.2. On average, the most frequently occuring idiom is *if H then C* and the least frequent *suppose H then C*. The list needs to be expanded and the development of more complex patterns and recursive patterns needs to be researched. With further tests on the ever-increasing arXMLiv sandboxes, we will obtain more statistics, which will give even more feedback about exactly which idiom patterns need to be added or improved.

In terms of performance, the Applicable Theorem Search system does not fall much behind MathWebSearch, with response times lower than the order of a second, as presented in Section 6.5. At the time of this thesis, the Applicable Theorem Search system keeps an index of the Connexions repository only, which is a rather limited resource, with only 431 files and 1714 theorem idioms (see Table 6.2). Unfortunately, the input problem of finding documents with correct Content MathML representation for the mathematics is the biggest obstacle standing in the way of both the MathWebSearch and Applicable Theorem Search systems. We can currently search the indexed theorem idioms on the basis of the mathematics through generalization search. In the future, we hope to employ idioms to identify universals in mathematical formulae from their natural language context, thus making the semantic dependancy between natural language and mathematics even stronger. The large number of idioms discovered in the Saarbrücken corpus (see Section 4.4) gives us a lot of hope for a truly large database of applicable theorems in the future, as soon as the LaMaPUn architecture is able to produce correct Content MathML and provide a solution to the input problem.

8.2 Future Work

This thesis presents a research prototype of a system meant to help scientists in their day-to-day research by providing easy access to deep semantic information. As such, the Idiom Spotter and Applicable Theorem Search systems described in the above chapters need still a lot of development and improvement before they can reach the desired standard. We will now list the most important work that needs to be carried out in the future, starting from the point where this thesis ends, looking at the two systems separately.

8.2.1 Idiom Spotter

Work items that still need to be done are listed below:

Improving Heuristic Pattern Matching Implementation The implementation of the method described in Section 3.3.2 needs to be perfected. At the moment, the implementation leaves the possibility of missing out on idioms with extra tags in the content. Extra cases need to be handled in the code, for the situation where the text and the math nodes are on several different DOM tree levels. At the moment, only the parent of the node with the keyword is analyzed. We should provide the heuristic for the case where the text node containing the keyword text might be surrounded by `<style>` tags or other content-irrelevant tags, in which case the parent of the text node would not offer the entire scope of the idiom. Also, a better solution needs to be found for documents containing special XML entities or different encodings. The current ad-hoc solution of manually downloading the DTDs, which I have taken for the Connexions corpus is not feasible for the future. Also, documents encoded in other formats than UTF8 (for example latin1) should be supported.

Comparing Idiom Spotting Methods The Heuristic Pattern Matching method provides only an initial low-level linguistic analysis. In order to tackle the problems outlined in Section 3.3.1, this method should be improved and other methods should also be analyzed. First, the current method should be improved by a more thorough analysis of the XMML processing, as mentioned above. Then, as a different method, by *semantically analyzing* the retrieved scope of the idiom, we can potentially identify the syntactical constructs which are more likely to fit a hypothesis or conclusion. For example, let us assume that a hypothesis always reduces to a noun phrase (NP). Then by obtaining the parse tree of the sentence before component extraction, we can extract only the NPs that fit the idiom pattern, thus eliminating the unneeded *filler* words. Another method to try is to extract the logical relations behind the sentence content of an idiom with Discourse Representation Structures [Cur07]. By looking at the structure of the DRS output of the Boxer [Box09] tool on a list of manually created idioms, the most oftenly occuring DRS patterns can be identified and the position where the hypothesis and the conclusion appear in such structures can be settled. These rules can be reused for idioms found in the analyzed texts. This is a method that extracts even more semantics from the text and it could be used for more complex idiom patterns and for reducing filler words.

Expanding Idiom Patterns. The list of idiom patterns needs to be improved. By looking at the already retreived idioms, we can determine whether more complex idiom patterns or just more simple patterns (like the ones used so far) are needed. A good approach might be to analyze the top five documents with most idioms found (the highest count is over 500 idioms in just one file) and see which language structures are missed out on. Other more complex patterns like *combined idioms* should be approached, as they would offer more involved semantics. Along the same lines, *context idioms* should be researched as a more accurate method of eliminating bound variables from the indexed theorems. The most simple first step in the field of idiom patterns is to add a lot of simple patterns to the idiom database.

Testing Field-specific Idioms An idea for the future is to create separate databases of idioms for different scientific fields. For example, we could have an idiom database for Chemistry, where a frequently occurring idiom pattern could be *reaction R rate equation T* or a mathematical definition database with many variations of the *define D as C* pattern. Since different fields employ a lot of different specific keywords, special field-specific idiom lists could be created.

Idiom Search Engine Apart from searching for idioms by instances of their mathematical content, we could provide a service that would offer search functionality by concept. This would be very useful when combined with definition idioms, as users could search for a particular concept and get its definition.

Test on More Corpora The system needs to be continuously tested, as the statistical test results provide the most important feedback mechanism for system improvement. More specialized databases need to be found and indexed. The arXMLiv group has provided the best support in providing input for the Idiom Spotter so far. Their converted corpus is not only finely categorized, but is also made up of quite recent scientific documents, which would provide a very up-to-date knowlede database.

8.2.2 Applicable Theorem Search

Since the Applicable Theorem Search system is completely dependant on the MathWebSearch and the Idiom Spotter systems, much of the future work that would improve its performance falls down to the individual components. However, there are still some points that belong to future improvement of the overarching system:

Find More Input The biggest problem of the Applicable Theorem Search system at the time of this thesis is the lack of proper mathematical input. The only corpus that is indexable is the Connexions repository, which offers a very limited collection of theorem idioms. The Saarbrücken corpus would be a perfect database for testing out the usefulness of the Applicable Theorem Search system, if the mathematics inside was in correct Content MathML format. However, until the transformation from LaTeX can output semantically correct Content MathML (or any other semantic format), it can not be used. A research effort in that direction can be seen in Section 7.3.

Improve Interface The Web Interface of the APPLICABLE THEOREM SEARCH system presented in this thesis is a prototype built up around the MATHWEBSEARCH GUI. For our system, a useful improvement would present clearer results, in a nicer formatting. Displaying the idiom pattern and how the hypotheses and conclusions fit into this pattern visually would make the result much easier to understand and use. Using colors to highlight the hypothesis and conclusion in the scope of the idiom would be a step in this direction. Also, we need to make sure that the resulting math formulae are properly displayed on the results page.

Context Idioms One topic that requires more research is context idioms. These patterns are specifically useful for the APPLICABLE THEOREM SEARCH, as they can decide on the status of universals or bound variables of mathematical symbols from the linguistic context. Introducing context idioms into the idiom database would add greater semantic depth to the analysis of universals and would also reduce the number of false positives caused by adding theorems with linguistic context-bound (but not mathematica formula-bound) variables.

Add Text Search The addition of text search capabilities to the APPLICABLE THEOREM SEARCH system through the `MaTeSearch` engine can be achieved at low implementation cost, given that the latter is already built up on MATHWEBSEARCH. This addition would empower the theorem search by narrowing down the size of the result set and providing a keyword idiom search engine.

Combined Idiom Category Database At the time of this thesis, the APPLICABLE THEOREM SEARCH system only provides storage and search capabilities for theorem idioms. In the future, a database structure should be created, which would be able to hold several different type of idiom categories, with a varying number of placeholder types. This would lead to a general idiom retrieval engine, similar to the theorem idiom search engine described in this thesis.

Compute Conditions One possible future improvement to the system could be a theorem conditions check. The system would identify the conditions from the hypotheses of a theorem result, compare them to the search query and signal to the user which conditions are met and which conditions still need to be met by the query term in order to apply the retrieved theorem. For example, if the user would search for $f(\sin(5))$ and the theorem would be *If f is continuous and $x \in \mathbb{N}$, then $f(sin(x)) = 0$*, the system would find that the first condition (f is continuous) still needs to be met by the user query before concluding that $f(\sin(5)) = 0$. This would require a much more thorough understanding of the mathematics in the hypothesis and conclusion and the complete semantic parsing of the math terms.

Conclusion

This thesis has presented a research project in the relatively new field of combined mathematics and natural language text processing. In the above exposition, I have described an attempt at structured information extraction from scientific texts with mathematical content. The main extraction "tools" have been natural language statements of a fixed structure, called *idioms*, concepts which inherently express a semantic relation between their components. Using these idioms as patterns for mixed mathematical and text discourse, I have shown how knowledge can be extracted, and categorized according to the semantic relation the idioms express. Focusing on one semantic relation as an example, I have proposed a system which provides search functionality for theorem idioms by their mathematical content, using a special mathematical term search engine.

The motivation behind my thesis, presented in Chapter 1 has been to provide a tool to help scientists advance their research by being able to search for theorems which they could apply to their mathematical computations. In Chapter 3, I have presented the theoretical background behind idioms, mathematical term search and the application of idiom patterns for information extraction. Based on this theory, I have implemented the Idiom Spotter utility, described in Chapter 4, which performs the actual retrieval from XML documents. In order to provide mathematical term search capabilities, I have looked into the existing MathWebSearch system and have given the practical analysis of its proposed extension in Chapter 5. Combining the MathWebSearch extension with the Idiom Spotter has led to the Applicable Theorem Search system (described in Chapter 6), which provides theorem idiom indexing and searching capabilities. The main problem encountered by the Applicable Theorem Search system and a potential solution to it are explained in Chapter 7. The evaluation given in Chapter 8 has shown the encouraging initial results of the extraction process and described planned future developments for the two systems introduced in this thesis.

In this thesis, based on assumptions about common linguistic practice in scientific documents, I have provided an exposition of the theory behind idiom spotting and how to combine it with mathematical search in order to bring applicable theorems to the user. As part of my research, I have also provided the systems that turn this theory into practice. The most important of the two, the Idiom Spotter has successfully proved its capabilities on mathematical corpora, confirming my theoretical assumptions about the use of idioms in texts with mathematical content. The Applicable Theorem Search system, although providing all the required idiom retrieval functionality, has not had a chance to prove its full potential yet. This is due to the limitations in proper mathematical input to the search engine caused by lack of user-authored Content MathML and by difficulties encountered in converting the valuable existing arXiv database to the desired representation. However, with the help of the promising LaMaPUn architecture, we hope to have semantically correct Content MathML in all converted arXMLiv documents soon. At that stage, we will have a huge corpus of theorems readily available for indexing and perfectly suited to test the limits of the Applicable Theorem Search system.

The research project described in this thesis presents only a small contribution to the analyses performed on large corpora with mathematical and natural language content. Many similar information extraction projects are being pursued inside the arXMLiv group, for the greater goal of enriching documents, which were originally written for presentation purposes, with structured semantics about their content. In this respect, idiom spotting can be generalized to the extraction of any semantic relation given implicitly by natural language. The more interesting research question discussed in this thesis is that of distinguishing natural language expressions which bring semantics to mathematical formulae. We have called these expressions context idioms and they are important because they tie together concepts from linguistics and logics, bringing much deeper semantics. The idea of context idioms can be pursued further to a point where the whole natural language scope of a mathematical formula can be interpreted formally, and transformed into a logical construction, thus offering a complete integrated math and text understanding.

Bibliography

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003.

[AGC+04] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A content based mathematical search engine: Whelp. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *TYPES*, volume 3839 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2004.

[AGLM99] A. A. Adams, Hanne Gottliebsen, S. A. Linton, and Ursula Martin. Vsditlu: a verifiable symbolic definite integral table look-up. In *CADE-16: Proceedings of the 16th International Conference on Automated Deduction*, pages 112–126, London, UK, 1999. Springer-Verlag.

[Anc07] Ştefan Anca. MaTeSearch, a combined math and text search engine. Bachelor's thesis, Computer Science, Jacobs University, Bremen, 2007.

[AP09] Ştefan Anca and Răzvan Paşcanu. Idiom spotter. Project Report as part of Computational Semantics of Natural Language, Jacobs University, Jan 2009.

[arX07] arXiv.org. Cornell University Library at `http://arxiv.org/`, seen May 2007.

[arX09] arXMLiv. Project home page at `http://arxmliv.kwarc.info/`, seen March 2009.

[ATS] Applicable theorem search. Demo at `http://betasearch.mathweb.org`.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.

[Box] C&C Boxer. Demo at `http://svn.ask.it.usyd.edu.au/trac/candc/wiki/Demo`.

[Box09] C&C Boxer. Project home page at `http://svn.ask.it.usyd.edu.au/trac/candc/wiki`, seen March 2009.

[Bri95] Eric Brill. Unsupervised learning of disambiguation rules for part of speech tagging, 1995.

[CCMr$^+$06] Ann Copestake, Peter Corbett, Peter Murray-rust, Advaith Siddharthan, Simone Teufel, and Ben Waldron. An architecture for language processing for scientific texts. In *Proceedings of the UK e-Science Programme All Hands Meeting 2006 (AHM2006), Nottingham, UK*, 2006.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[CMR06] Peter Corbett and Peter Murray-Rust. High-throughput identification of chemistry in life science texts. In Michael R. Berthold, Robert C. Glen, and Ingrid Fischer, editors, *CompLife*, volume 4216 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006.

[CNX09] Connexions. Project home page at `http://cnx.rice.edu/`, seen February 2009.

[Cop07] Ann Copestake. Applying robust semantics. In *Proceedings of PACLING 2007, the 10th Conference of the Pacific Association for Computational Linguistics, Melbourne, Australia*, 2007.

[coq09] The Coq proof assistant. `http://coq.inria.fr/`, seen March 2009.

[CTW06] Ann Copestake, Simone Teufel, and Benjamin Waldron. Flexible interfaces in the application of language technology to an escience corpus. In *Proceedings of the 4th UK E-Science All Hands Meeting*, 2006.

[Cur07] James R. Curran. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the Demonstrations Session of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 29–32, 2007.

[DLM05] Digital Library of Mathematical Functions. Project home page at `http://dlmf.nist.gov/`, April 2005.

[FC99] Richard J. Fateman and Eylon Caspi. Parsing tex into mathematics. *SIGSAM Bull.*, 33(3):26, 1999.

[Fre91] Free Software Foundation. Gnu general public license, 1991.

[GJA$^+$09] Deyan Ginev, Constantin Jucovschi, Ştefan Anca, Mihai Grigore, Cătălin David, and Michael Kohlhase. An architecture for linguistic and semantic analysis on the arxmliv corpus. In *Proceedings of AST'09*, 2009.

[Goo09] Google search engine. `http://www.google.com`, seen August 2009.

[Gra96] Peter Graf. *Term Indexing*, volume 1053 of *Lecture Notes in Computer Science*. Springer, 1996.

[HC06] Aurelie Herbelot and Ann Copestake. Acquiring ontological relationships from wikipedia using rmrs. In *ISWC 2006 Workshop on Web Content*, 2006.

[KAJ$^+$08] Michael Kohlhase, Ştefan Anca, Constantin Jucovschi, Alberto González Palomo, and Ioan Şucan. MathWebSearch 0.4, a semantic search engine for mathematics. Submitted to MKM-08, 2008.

[Kam95] Hans Kamp. Discourse representation theory. In J. Verschueren, J.-O. Östman, and J. Blommaert, editors, *Handbook of Pragmatics*, pages 253–257. Benjamins, 1995.

[KS06] Michael Kohlhase and Ioan Sucan. A search engine for mathematical formulae. In Jacques Calmet, Tetsuo Ida, and Dongming Wang, editors, *AISC*, volume 4120 of *Lecture Notes in Computer Science*, pages 241–253. Springer, 2006.

[KWA09] Kwarc. `http://kwarc.info`, seen Feb. 2009.

[LaM09] LaMaPUn. Group home page at `http://kwarc.info/projects/kwarcnlp/`, seen April 2009.

[LaT09] Latex. Project home page at `www.latex-project.org`, seen August 2009.

[Lin09] Link Parser. Project home page at `http://www.link.cs.cmu.edu/link/`, seen March 2009.

[Mat] MathWebSearch a semantic search engine. Demo at `http://search.mathweb.org`.

[Mat09] Math Web Search. `http://kwarc.info/projects/mws/`, seen Feb. 2009.

[Mil09] Bruce Miller. `LaTeXML`: A LaTeX to xml converter. Web Manual at `http://dlmf.nist.gov/LaTeXML/`, seen March2009.

[OMD] OMDoc. web page at `http://omdoc.org`.

[Ope09] OpenMath Home. `http://www.openmath.org/`, seen May 2009.

[Pal06] Alberto González Palomo. Sentido: an authoring environment for OMDoc. In OMDoc – *An open markup format for mathematical documents [Version 1.2]*, number 4180 in LNAI, chapter 26.3. Springer Verlag, 2006.

[PVLA07] Andrei Paskevich, Konstantin Verchinine, Alexander Lyaletski, and Anatoly Anisimov. Reasoning inside a formula and ontological correctness of a formal mathematical text. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Calculemus/MKM 2007 — Work in Progress*, number 07-06 in RISC-Linz Report Series, University of Linz, Austria, pages 77–91, Hagenberg, Austria, June 2007.

[RCC$^+$08] C.J. Rupp, Ann Copestake, Peter Corbett, Peter Murray-Rust, Advaith Siddharthan, Simone Teufel, and Benjamin Waldron. Language resources and chemical informatics. In European Language Resources Association (ELRA), editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008.

[Sci08] HTTP://WWW.SCIBORG.ORG.UK. Extracting the Science from Scientific Publications project at `http://www.sciborg.org.uk`, seen November 2008.

[SK08] Heinrich Stamerjohanns and Michael Kohlhase. Transforming the ar χ iv to xml. In *Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics*, pages 574–582, Berlin, Heidelberg, 2008. Springer-Verlag.

[ST93] C Daniel Sleator and Davy Temperley. Parsing english with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[The06] The Apache Software Foundation. Lucene. `http://lucene.apache.org/`, 2000–2006.

[VLPA08] Konstantin Verchinine, Alexander Lyaletski, Andrei Paskevich, and Anatoly Anisimov. On correctness of mathematical texts from a logical and practical point of view. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics, AISC/Calculemus/MKM 2008*, volume 5144 of *Lecture Notes in Computer Science*, pages 583–598, Birmingham, United Kingdom, July 2008. Springer.

[W309] MathML Software - Authoring Systems. Page at `http://www.w3.org/Math/Software/mathml_software_cat_authoring.html`, seen Aug 2009.

[W3C07] W3 consortium, seen February 2007. webpage at `http://www.w3.org`.

[web] The size of the world wide web. Seen at `http://www.worldwidewebsize.com/`.

[Wol99] Stephen Wolfram. *The Mathematica book (5th edition).* Cambridge University Press, 1999.

[Wol09] Wolfram functions. web page at `http://functions.wolfram.com`, seen August 2009.

[xma08] Formelsammlungen: Mathematik, physik, technik, und finanzmathematik, seen April 2008.

[You06] Abdou Youssef. Roles of math search in mathematics. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI, pages 2–16. Springer Verlag, 2006.

[Zin04] Claus Zinn. *Understanding Informal Mathematical Discourse.* PhD thesis, Technischen Fakultät der Universität Erlangen-Nürnberg, 2004.

Appendix A

Appendix

A.1 Running Example

Case 1: If given a recurrence relation of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

with $a \geq 1$, $b > 1$ and $f(n) = \mathcal{O}\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

A.2 Running Example in XHTML format (fragment)

```
<p class="p"><span style="" class="text bold">Case 1:</span> If given a recurrence relation of the form</p>
  <m:math display="block">
    <m:semantics>
      <m:mrow>
        <m:mrow>
          <m:mi>T</m:mi>
          <m:mo> </m:mo>
          <m:mfenced open="(" close=")">
            <m:mi>n</m:mi>
          </m:mfenced>
        </m:mrow>
        <m:mo>=</m:mo>
        <m:mrow>
          <m:mrow>
            <m:mi>a</m:mi>
            <m:mo> </m:mo>
            <m:mi>T</m:mi>
            <m:mo> </m:mo>
            <m:mfenced open="(" close=")">
              <m:mfrac>
                <m:mi>n</m:mi>
                <m:mi>b</m:mi>
```

```
                  </m:mfrac>
                </m:mfenced>
              </m:mrow>
              <m:mo>+</m:mo>
              <m:mrow>
                <m:mi>f</m:mi>
                <m:mo> </m:mo>
                <m:mfenced open=”(” close=”)”>
                  <m:mi>n</m:mi>
                </m:mfenced>
              </m:mrow>
            </m:mrow>
          </m:mrow>
          <m:annotation−xml encoding=”MathML−Content”>
            <m:apply>
              <m:eq/>
              <m:apply>
                <m:times/>
                <m:ci>T</m:ci>
                <m:ci>n</m:ci>
              </m:apply>
              <m:apply>
                <m:plus/>
                <m:apply>
                  <m:times/>
                  <m:ci>a</m:ci>
                  <m:ci>T</m:ci>
                  <m:apply>
                    <m:divide/>
                    <m:ci>n</m:ci>
                    <m:ci>b</m:ci>
                  </m:apply>
                </m:apply>
                <m:apply>
                  <m:times/>
                  <m:ci>f</m:ci>
                  <m:ci>n</m:ci>
                </m:apply>
              </m:apply>
            </m:apply>
          </m:annotation−xml>
        </m:semantics>
      </m:math>
<p class=”p”>, with
  <m:math display=”inline”>
    <m:semantics>
      <m:mrow>
        <m:mi>a</m:mi>
        <m:mo>&leq;</m:mo>
        <m:mn>1</m:mn>
      </m:mrow>
      <m:annotation−xml encoding=”MathML−Content”>
         <m:apply>
           <m:geq/>
           <m:ci>a</m:ci>
           <m:cn>1</m:cn>
         </m:apply>
      </m:annotation−xml>
     </m:semantics>
  </m:math>,
  <m:math display=”inline”>
    <m:semantics>
      <m:mrow>
         <m:mi>b</m:mi>
         <m:mo>&gt;</m:mo>
```

```
              <m:mn>1</m:mn>
            </m:mrow>
            <m:annotation−xml encoding="MathML−Content">
              <m:apply>
                <m:gt/>
                <m:ci>b</m:ci>
                <m:cn>1</m:cn>
              </m:apply>
            </m:annotation−xml>
          </m:semantics>
        </m:math> and <m:math display="inline">
```

Index