| | |
|---|---|
| TITLE | A proof-theoretic approach to mathematical knowledge management |
| AUTHOR | Aboul-Hosn, Kamal |
| DEGREE | PhD |
| SCHOOL | CORNELL UNIVERSITY |
| DATE | 2007 |

# A PROOF-THEORETIC APPROACH TO MATHEMATICAL KNOWLEDGE MANAGEMENT

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Kamal Aboul-Hosn

January 2007

## BIOGRAPHICAL SKETCH

Kamal Aboul-Hosn, son of Sydney and Hussein Aboul-Hosn, grew up in Bellefonte, Pennsylvania. With an interest in computers, he entered the Pennsylvania State University in August 1998. While there, he worked on several projects, including his honors thesis on programming with private state, under the guidance of John Hannan, and a parser generator for $\lambda$Prolog, under Dale Miller. He also served as a Technology Learning Assistant, teaching more than twenty faculty members how to effectively use technology in their classrooms. Kamal was graduated from Penn State with an honors B.S. in Computer Science and minor in Mathematics in December 2001.

In August 2002, Kamal entered the Ph.D. program in Computer Science at Cornell University. What started as a final project for a class turned into his thesis work on the formal representation of mathematical knowledge, under the guidance of Dexter Kozen. He has also worked on program verification using Kleene algebra with tests. As a part of the Graduate Student School Outreach Project, Kamal taught an eight-week mini-course on artificial intelligence to local high school students. Kamal was graduated from Cornell University with a Ph.D. in Computer Science and a minor in Economics in January 2007.

Kamal is an avid drummer and photographer. He is also the developer of *Radar In Motion*, an animated weather map widget for Apple's Mac OS X *Dashboard*.

## ACKNOWLEDGEMENTS

"There are things we don't say often enough. Things like what we mean to one another. All of you mean a lot to me. I just want you to know that." –Bill Adama

This thesis would not have been possible without the help of many people. First and foremost, I have to thank my family. I am who I am today because of the love and guidance of my parents, Hussein and Sydney Aboul-Hosn. My sister, Hannah, who is one of the greatest people I know, has been there through everything. I am so proud of you. I must also mention my grandparents, Caroline and Orlen Rice, the latter of whom instilled in me a love of science and engineering at a very young age. You are greatly missed, grandypa. And finally, Sittee, with whom I share a deep bond that words will not do justice. I love all of you.

I cannot possibly convey the gratitude I feel toward Dexter Kozen, my advisor, colleague, and friend. He has been a constant source of inspiration academically, musically, and personally. Many thanks for helping so many of my dreams come true. Vicky Weissman once astutely noted that "for an advisor, you have to pick someone you can see yourself becoming in ten years." I'm very comfortable with the idea of ending up like Dexter.

I also want to thank those who have made my years here at Cornell University so memorable. I have found many lifelong friends in my four and a half years here. I have been extremely fortunate to know people such as Milind Kulkarni and Ganesh Ramanarayanan, with whom I will always remember skipping stones at Stewart Park my first week in Ithaca–my first realization that life here was going to be all right. Siggi Cherem and Chethan Pandarinath have provided many laughs and memorable moments. I look forward to seeing all of you again, even as our paths now diverge. I can't forget those with whom I played music in my time here, including Dexter, Joel Baines, and Steve Chong. To Becky Stewart, Stephanie Meik, and Kelly Patwell, thank you for all of the guidance, assistance, and fun times. I must also mention Polly Israni, Nikita Kuznetsov, and Terese Damhøj Andersen.

There are several people I've known from earlier in my life I'd like to thank. No one can have better friends than BNG Spicer and Lee Armstrong, who are like brothers to me, and my comrade, Justin Miller, whom I have known for almost 20 years. I am also fortunate to have worked with John Hannan and Dale Miller in my time at Penn State. Several others have also had a deep and meaningful impact on my life, including Matt Weirauch, Sean Fox, and Princess Laura Murphy.

My sincerest thank you to all of you who have supported me, helped me, challenged me, excited me, who have made my life what it is today.

## TABLE OF CONTENTS

**1 Introduction** **1**

**2 Related Work** **9**
- 2.1 Proof Reuse . . . 9
  - 2.1.1 Proof Analogy . . . 10
  - 2.1.2 Proof Abstraction . . . 13
  - 2.1.3 Applications of Proof Reuse . . . 16
- 2.2 Library Organization . . . 18
  - 2.2.1 Representing Mathematics for Wide Dissemination . . . 20
  - 2.2.2 Creating a Large Mathematical Knowledge Base . . . 23
  - 2.2.3 Proof Organization in Automated Theorem Provers . . . 24
  - 2.2.4 Extracting Libraries from Automated Theorem Provers . . . 28
- 2.3 Tactics . . . 31
  - 2.3.1 Theoretical Developments in Tactics . . . 31
  - 2.3.2 The Reuse of Tactics . . . 36

**3 Publication-Citation** **38**
- 3.1 Motivation: A Classical Proof System . . . 38
- 3.2 Explicit Library Representation . . . 41
- 3.3 An Example . . . 44

**4 KAT-ML** **48**
- 4.1 Preliminary Definitions . . . 51
  - 4.1.1 Kleene Algebra . . . 51
  - 4.1.2 Kleene Algebra with Tests . . . 52
  - 4.1.3 Schematic KAT . . . 54
- 4.2 Description of the System . . . 55
  - 4.2.1 Rationale for an Independent Implementation . . . 55
  - 4.2.2 Overview of KAT-ML . . . 57
  - 4.2.3 Representation of Proofs . . . 58
  - 4.2.4 Citation . . . 59
  - 4.2.5 An Extended Example . . . 62
  - 4.2.6 Heuristics and Reductions . . . 64
  - 4.2.7 Proof Output and Verification . . . 67
  - 4.2.8 SKAT in KAT-ML . . . 67
  - 4.2.9 A Schematic Example . . . 73
- 4.3 Conclusions . . . 79

**5 Hierarchical Math Library Organization** **82**
5.1 A Motivating Example . . . . . 83
5.2 Proof Representation . . . . . 85
5.3 Proof Rules . . . . . 89
5.3.1 Rules for Manipulating Proof Tasks . . . . . 91
5.3.2 Rules for Manipulating Proof Terms . . . . . 94
5.4 A Tree Structure Represention of Proof Terms . . . . . 97
5.5 Proof Term Manipulations on Trees . . . . . 98
5.6 A Constructive Example . . . . . 100
5.7 Conclusions . . . . . 107

**6 User Interfaces** **108**
6.1 Proofs Represented as Graphs . . . . . 108
6.2 Current User Interfaces . . . . . 110
6.3 A Proof-Theoretic User Interface . . . . . 115

**7 Tactics** **119**
7.1 A Motivating Example . . . . . 121
7.2 Tactic Representation . . . . . 122
7.3 A Constructive Example . . . . . 130
7.4 Conclusions . . . . . 134

**8 Future Work** **136**
8.1 Proof Refactorization . . . . . 136
8.2 Tactic Type Systems . . . . . 137
8.3 Implementation . . . . . 138
8.4 Online Library Sharing . . . . . 139

**9 Conclusions** **141**

**A Library Organization Soundness** **143**

**References** **161**

## LIST OF FIGURES

1.1 A schematic view of the branches of mathematics [101] . . . . . . . 2
1.2 Number of mathematics journals [3] . . . . . . . . . . . . . . . . . . 3
1.3 Distribution of publication dates for computer science papers [2] . . 4

3.1 Rules for a classical proof system . . . . . . . . . . . . . . . . . . . 40
3.2 Typing rules for proof terms . . . . . . . . . . . . . . . . . . . . . . 41
3.3 Annotated proof rules . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.4 Proof Rules for Basic Theorem Manipulation . . . . . . . . . . . . . 42

4.1 KAT-ML main window . . . . . . . . . . . . . . . . . . . . . . . . . 58
4.2 KAT-ML first-order window . . . . . . . . . . . . . . . . . . . . . . 69
4.3 Proof steps for theorem from [78] . . . . . . . . . . . . . . . . . . . 72
4.4 Generated LaTeX output . . . . . . . . . . . . . . . . . . . . . . . . 73
4.5 Schemes $S_{6A}$ and $S_{6E}$ . . . . . . . . . . . . . . . . . . . . . . . 74
4.6 Scheme equivalence theorem . . . . . . . . . . . . . . . . . . . . . . 75
4.7 Translation table for scheme proof . . . . . . . . . . . . . . . . . . 76
4.8 Proof steps for $tf = tf(C \leftrightarrow D)$ . . . . . . . . . . . . . . . . . . . 78
4.9 Proof term for $tf = tf(C \leftrightarrow D)$ . . . . . . . . . . . . . . . . . . . 79
4.10 Proof steps for $(bc)^*a \leq a(bc)^*$ . . . . . . . . . . . . . . . . . . . 80

5.1 Typing rules for proof terms . . . . . . . . . . . . . . . . . . . . . . 87
5.2 Typing rules for proof library . . . . . . . . . . . . . . . . . . . . . 91
5.3 Rules for manipulating proof tasks . . . . . . . . . . . . . . . . . . 92
5.4 Rules for manipulating the proof library . . . . . . . . . . . . . . . 93
5.5 Rules for manipulating proof terms in $\mathcal{C}$ . . . . . . . . . . . . . . . 95
5.6 Translation between proof terms and proof trees . . . . . . . . . . 99
5.7 (5.7) as a proof tree . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.8 (5.8) as a proof tree . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.9 **(promote)** as a tree manipulaiton . . . . . . . . . . . . . . . . . . 101
5.10 **(push)** and **(pull)** as tree manipulaitons . . . . . . . . . . . . . . 101
5.11 **(generalize)** and **(specialize)** as tree manipulaitons . . . . . . . . 102
5.12 **(split)** and **(merge)** as tree manipulaitons . . . . . . . . . . . . . 103

6.1 An example theory layout in HOL [106] . . . . . . . . . . . . . . . 111
6.2 Replaying a proof in Isar in Proof General [7] . . . . . . . . . . . . 113
6.3 Hoare logic rules arranged in hierarchical fashion . . . . . . . . . . 116
6.4 Right-click options . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

7.1 Typing rules for new proof terms . . . . . . . . . . . . . . . . . . . 125
7.2 Proof Rules for Basic Theorem Manipulation . . . . . . . . . . . . . 127
7.3 Proof Rules for Tactics . . . . . . . . . . . . . . . . . . . . . . . . . 129

# Chapter 1

# Introduction

Mathematics is a field that is important in our day-to-day lives. From a very young age, our children are taught the fundamentals of arithmetic. As they progress through middle school and high school, students learn algebra, geometry, trigonometry, and even calculus. In college, the mathematical knowledge we impart to these students becomes more specialized. Economics students learn about derivatives and their use in reasoning about changes in markets. Future physicists use calculus to model the properties of matter and energy.

Those who continue on to advanced courses and graduate degrees learn of the beautiful abstractions that provide a common basis for much of the mathematics they knew most of their lives. It is at this point that they truly understand the intricate hierarchy that binds the entire field of mathematics and all its applications, such as the one see in Figure 1.1. Within each area, the hierarchy gets more specific, branching into many subtopics. For example, the area of differential geometry breaks down further into the geometry of curves, the geometry of surfaces, Riemannian geometry, and several others.

Each part of this hierarchy has its own set of definitions, theorems, axioms, and proofs that are fundamental to that area. Often, theorems in subareas are specialized versions of those that appear higher up in the hierarchy. It may be the case that the proof of a specialized theorem is easier in a subarea because one can take advantage of certain properties that are not true of the more general area. As an example, many mathematical structures with structure-preserving maps including sets, monoids, and rings can be viewed more generally as categories with

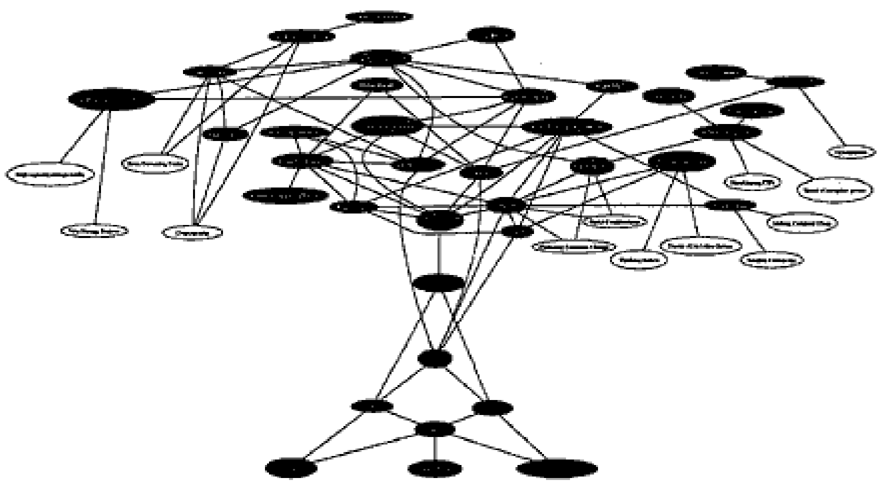

Figure 1.1: A schematic view of the branches of mathematics [101]

morphisms. Some of the properties of the operations on these structures are results regarding morphisms in category theory.

The teaching of mathematics starts with simple, specific concepts and then moves to more general concepts that encompass those already learned as one gets more advanced. Research in mathematics can work in both directions: one starts from more specific ideas and generalizes them; or, one takes general ideas and specializes them to work in a specific instance. It is not always clear which way one is going because the area of mathematics is so large; one may write a paper establishing some new theorems in a subarea of mathematics, only to discover that the work is closely related to or a special case of work in a more general area of which the author was not aware.

The relationships in mathematics are becoming richer as the body of mathematical knowledge is constantly increasing and changing. The number of mathematics journals has continued to increase over the last century and a half, as demonstrated in Figure 1.2. These journals have continued to become more specific in their topics, indicating that the study of mathematics is becoming more advanced.

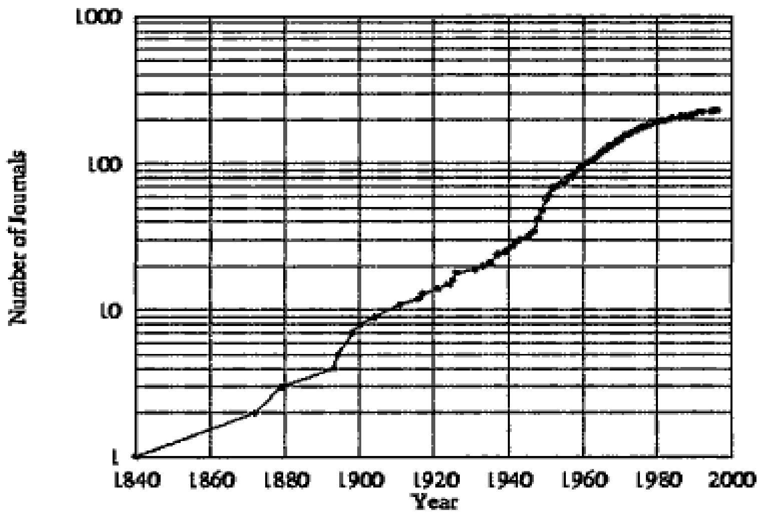

Figure 1.2: Number of mathematics journals [3]

If we look specifically in the field of computer science, where mathematics plays a prominent role, we see a dramatic increase in the number of papers published over the years. For example, the Digital Bibliography and Library Project (DBLP), which provides bibliographic information from papers in major computer science conferences and journals, has seen a dramatic increase in papers, demonstrated in Figure 1.3.

With an increase in the amount and complexity of the information, the organization of mathematical knowledge becomes more important and more difficult. This emerging area of research is referred to as *mathematical knowledge management* (MKM). As summarized by Buchberger, one of the organizers of the first Mathematical Knowledge Management Workshop, the phrase "mathematical

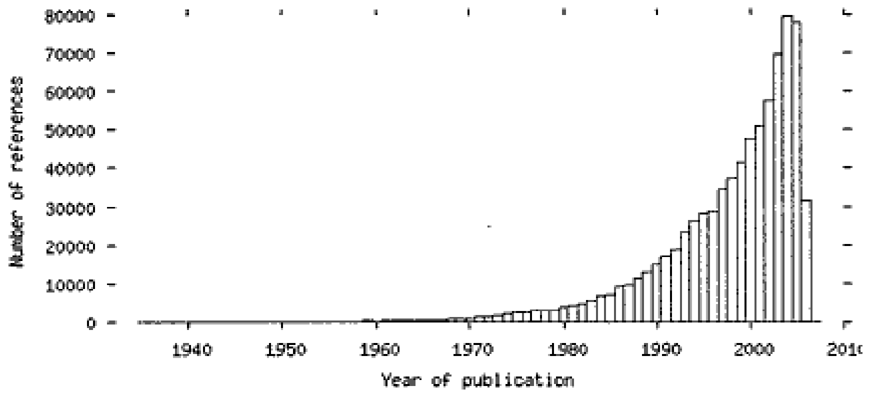

Figure 1.3: Distribution of publication dates for computer science papers [2]

knowledge management" should be parsed as (mathematical knowledge) management as opposed to mathematical (knowledge management), i.e., examining the problem of organizing and disseminating mathematical knowledge [20]. He goes on to summarize the primary issues in the field:

- How do we retrieve mathematical knowledge from existing and future sources?
- How do we build future mathematical knowledge bases?
- How do we make the mathematical knowledge bases available to mathematicians?

At least part of MKM's development has come from the growing power of and interest in *automated theorem proving*, using computer programs to prove mathematical theorems. Using automated theorem provers to find the proofs for theorems offers several advantages. First of all, much of the process can be automated through the use of heuristics called *tactics* and *tacticals*. These heuristics perform basic steps of reasoning, including search for the correct steps to take.

With constantly increasing computer power, more efficient tactics, and research into new search strategies, the portion of the theorem-proving process that can be automated continues to increase.

The primary contribution of automated theorem proving that is relevant to MKM is the formalization of mathematics. A proof written by hand by a mathematician tends to have some steps that are informal or appeal to some intuition on the part of the reader. We even see phrases like "the proof is trivial" or "this step is obvious" in proofs in papers and textbooks. In contrast, a proof produced by a computer program must be rigorous, with every detail justified by a step of reasoning that follows in the domain of the theorem being proven; there is no such thing as "trivial" or "obvious" for an automated theorem prover.

The body of formalized mathematics has continued to increase, with results spanning all major branches of mathematics. As with any large body of information, there is a desire to organize all of these formal theorems and proofs into a digital library. We can then take advantage of the formal structure of these proofs for research and teaching. From a research perspective, we can use the formal library to find theorems useful in a proof we are working on or to discover related theorems based on common proof steps. For teaching, a formalized library of mathematics provides a structured way to organize one's presentation of complex theorems and proofs related to one another.

The basis of any formalized structure for mathematics is a library of proofs and theorems. Large libraries do exist in the automated theorem provers. However, these libraries are usually implemented at the system level, meaning they are not defined with the same level of formalism as the proofs themselves, which rely on a strong underlying proof theory with rules for their creation. In the same way a proof-theoretic approach can formalize the steps of a proof, we want a proof-theoretic approach that can formalize the relationships between proofs in a library. The library should have the following properties:

1. **Independence** The library should be independent of the underlying logic for which proofs are being done; we should be able to organize proofs for any area of mathematics.

2. **Structure** The formal layout of the library should reflect relationships between theorems. In other words, if we regard a proof to be a lemma used within a larger proof, then the proofs should be such that the relationship is captured inherently in the structure.

3. **An underlying formalism** Proof-theoretic rules should be the basis of manipulating the library. They should formally define the operations of adding a proof to the library, removing a proof from the library, and using one proof in another. These rules should be defined at the same level as the rules used for creating proofs.

4. **Adaptability** The organization of the proofs in the library should be able to change based on the desire to highlight different relationships. For example, one may want to change the structure to group different theorems based on a certain set of lemmas they all use. Changes should be formally described by rules.

5. **Presentability** The formal library needs itself either to be easily read by humans or to be translatable into a format that can be read by humans. The format should reflect the structure of the library and, ideally, be alterable

in a way controlled by the underlying proof-theoretic rules for adapting the library.

The libraries in all of the popular automated theorem provers including Coq [105], NuPRL [71], Isabelle [108], and PVS [89] exhibit the first property. The second property, structure, is found in theorem provers in an informal way. One can declare formulas to be lemmas instead of theorems, however, no distinction is made by the systems themselves. Progress has been made, particularly in Isabelle, toward providing some more structure to the library of theorems.

Properties 3 and 4 are not exhibited by any of the popular theorem provers. As stated, the library is a system-level construct separated from the underlying logic governing the creation of proofs. Therefore, no formalism controls the library. Combined with the fact that there is no structure inherent in the library, adaptability is extremely limited.

Presentability has been addressed by the theorem prover community by taking existing libraries from automated theorem provers and transforming them into a readable format, usually for presentation on the Internet.

In this thesis, we present a proof-theoretic approach to mathematical knowledge management that exhibits all five desired properties. In Chapter 2, we discuss previous work from several aspects of the problem, including proof reuse and library organization. In Chapter 3, we set the basis for a library that exhibits properties 1 and 3 by discussing a *publish-cite* system presented by Kozen and Ramanarayanan [68]. We look at an implementation of this library in an interactive theorem prover for Kleene algebra with tests [63] in Chapter 4. We satisfy properties 2, 4, and 5 by formally defining a hierarchical structure for the mathematical library in Chapter 5. In Chapter 6, we discuss user interfaces for theorem provers and present a prototype theorem prover for Kleene algebra with tests that presents the library of theorems in an intuitive, structured format. We extend the formalism of the library to include tactics, allowing them to be treated at the same level as proofs and the library itself, in Chapter 7. Finally, we present some future directions for the work and conclusions in Chapters 8 and 9.

# Chapter 2

# Related Work

The development of formal methods for proof representation and theorem proving has both a rich history and a community that remains active. Much of this work is in automated theorem provers such as Coq [105], NuPRL [71], Isabelle [108], and PVS [89].

The cores of these systems, where issues such as proof representation and the underlying proof logic must be considered, have been well studied and established. However, there are other distinctive characteristics that are paramount to the development of these systems that continue to be important research questions, including proof reuse, proof library representation, and proof tactics. These issues have a serious impact on system usability, both for presenting information to a user and for implementing the system efficiently. We examine the work related to each one of these considerations in detail.

## 2.1 Proof Reuse

Reusing proofs is important for several reasons. The most obvious is that one does not want to have to perform steps repeatedly when they can be done once and referred to later. From the perspective of an automated theorem prover, time is saved in reusing completed proof steps. The other important reason for proof reuse is that discovering proofs with the same steps helps to establish relationships between theorems, including some that might otherwise go unnoticed.

Carbonell succinctly states the four aspects of problem solving that are relevant to proof reuse, where we transfer information from one proof, called the *source* *proof*, to another proof, called the *target proof* [25]:

1. How does one define similarity in proofs?
2. What knowledge is transferred from the source proof to the target proof?
3. How is this transfer accomplished?
4. How does one choose related source proofs given a target proof?

### 2.1.1 Proof Analogy

A popular method for proof reuse initially explored by mathematicians and artificial intelligence researchers is the idea of *proof analogy*, which tries to map steps from a source proof into steps in a target proof using hints in the relationship between the source theorem and target theorem.

Early work by Kling [56] and Munyer [87] focused on using the source proof to find inference rules that would be relevant for the target proof. Kling's technique can find analogous inference rules for the target proof, but is not designed to use the structure of the source proof to guide the decisions made in the use of these rules. Munyer's work, however, is able to use the order of inference rules in the source proof in order to guide the target proof.

A severe limitation of these approaches is that they define similarity in a purely syntactic sense; syntactic analogy can only discover, for instance, that the proof "if $x$ and $y$ are even, then $x*y$ is even" is related to the proof "if $x$ and $y$ are odd, then $x*y$ is odd." Several others explored other notions of analogy in order to make the technique more powerful, both in finding similar theorems and in applying their proofs.

Carbonell worked on *transformational analogy* and *derivational analogy* in the context of general artificial intelligence problem solving techniques [25]. Carbonell's work dealt primarily with the third element in our list above; but his work also has implications for choosing related theorems and proofs. We talk about his work as it would be applied to theorem proving. Both transformational analogy and derivational analogy attempt to solve a proof by looking at sequences of proof steps that were successful in some previous proof and using them in the target proof. They require the storage of previously completed theorems and their proofs.

*Transformational analogy* looks for similarity in the statements of theorems, copies the proof for a relevant source theorem, and attempts to adapt the proof to solve the target theorem. The notion of similarity here is vague; it could be as simple as syntactic matching or could use some more complicated metric defined by a user.

In contrast, *derivational analogy* matches source and target proofs instead of theorems. One starts searching for steps in the target proof and then looks for a source proof that has a similar pattern of search. The search procedure for the source proof is then copied to the target proof and used to find a solution. Derivational analogy requires that the proof steps that failed be stored with a proof, in addition to the steps that succeeded. By using the steps from the source proof, one creates a *proof plan*, which guides the steps of searching for a proof of the target theorem [22].

Both of these techniques can be inefficient given a complex similarity metric and large library of previous proofs. The library becomes particularly large when using derivational analogy. Cabonell applied his techniques primarily to natural language processing and looked at the library of knowledge in that context. Nevertheless, it is obvious that these techniques applied to proof reuse require a well organized library of theorems and proofs.

Melis and Whittle have worked extensively on applying analogy to inductive proofs, particularly for the proof planner *$CL^AM$* [79, 110, 81, 80, 82]. They split analogy into two forms: *internal analogy*, which looks for similar subgoals within a single proof, and *external analogy*, which looks for similar theorems outside the context of the current proof. Jamnik demonstrated that Melis and Whittle's technique applies to non-inductive proofs as well [49].

*Internal analogy* tries to make the search for a proof more efficient by reducing the number of calls to *$CL^AM$*'s *critic*, which attempts to revise terms on which induction is being performed when the inductive proof can no longer make progress. When *$CL^AM$* needs to choose a term on which to perform induction, its analogy system suggests one based on the terms chosen by previous calls to the critic. The suggestions, if successful, prevent the critic from having to search for a term on which to perform induction and prevent the system from performing inductive proofs that will inevitably fail. The use of internal analogy has been able to produce measurable reductions in the time it takes to perform an inductive proof in *$CL^AM$*.

*External analogy* also attempts to reduce the need for search in *$CL^AM$*. Melis and Whittle implemented an analogy procedure, ABALONE, on top of *$CL^AM$*. ABALONE attempts to find a second-order mapping from source theorems to target theorems. Theorems are represented as syntactic trees in which paths containing existentially quantified variables and induction variables, called *rippling paths*, are marked. Completed theorems–including decisions made in the planning of the theorem's proof, called *justifications*–are maintained in a library. If a useful