

University of Edinburgh  
School of Informatics

The Use of Data-Mining for the Automatic  
Formation of Tactics

Thesis Proposal

Hazel Duncan

February 3, 2004

**Abstract:** The aim of this project is to evaluate the applicability of data-mining techniques to the automatic formation of tactics from large corpuses of proofs.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Review of Existing Literature</b>	<b>3</b>
2.1	Proof Planning Background . . . . .	3
2.1.1	$\lambda$ Clam and $\Omega$ mega . . . . .	4
2.2	Previous Learning Work . . . . .	5
2.2.1	Pre-condition Analysis . . . . .	5
2.2.2	Learning using Markov Models . . . . .	5
2.2.3	Learning Proof Methods . . . . .	6
2.2.4	Random Fields . . . . .	6
2.2.5	Learning Heuristic Control . . . . .	7
2.2.6	Proof Reuse and Simulation of Human Learning . . . . .	7
2.3	Previous Work on Genetic Programming . . . . .	8
2.3.1	Koza . . . . .	8
2.3.2	Evolutionary Programming . . . . .	9
<b>3</b>	<b>Description of Central Ideas</b>	<b>11</b>
3.1	Choosing the Corpus . . . . .	12
3.1.1	Isabelle as a Candidate . . . . .	12
3.1.2	Mizar as a Candidate . . . . .	13
3.2	Data-Mining the Proof Corpuses . . . . .	14
3.3	Genetic Programming . . . . .	15
<b>4</b>	<b>Work so Far</b>	<b>17</b>
4.1	Extracting the Data . . . . .	17
4.2	Data-Mining Software . . . . .	17
4.3	Design Specific Software . . . . .	18
4.4	Feasibility Tests . . . . .	20
<b>5</b>	<b>Future Targets</b>	<b>21</b>
5.1	Milestones . . . . .	21
5.2	Dependencies . . . . .	22
5.3	Deliverables . . . . .	23
<b>6</b>	<b>Likely Outcome</b>	<b>25</b>
	<b>Bibliography</b>	<b>27</b>



# 1. Introduction

This proposal draws on one of the more traditional themes of Artificial Intelligence: automated deduction. Automated deduction has traditionally been thought of with great optimism, unfortunately, the huge search spaces involved in searching for correct proofs has meant that fully automated theorem provers are not as advanced as it was once thought they would be by this time. For example, Newell and Simon claimed that a computer would “discover and prove an important new mathematical theorem” by January 1st 1968 [32]. The introduction of tactics helped the field of automated deduction by guiding search, this project aims to build upon that success by implementing a method to allow proof plans to be formed automatically.

Proof planning is a technique developed at Edinburgh for guiding the search for a proof in automated deduction. A proof plan is an outline or plan of a proof. To prove a conjecture, proof planning first constructs the proof plan for a proof and then uses it to guide the construction of the proof itself.

Common patterns in proofs are identified and represented in computational form as general-purpose tactics, i.e. programs for directing the proof search process. These tactics are then formally specified with methods using a meta-language. Standard patterns of proof failure and appropriate patches to the failed proofs attempts are represented as critics. To form a proof plan for a conjecture the proof planner reasons with these methods and critics. The proof plan consists of a customised tactic for the conjecture, whose primitive actions are the general-purpose tactics. This customised tactic directs the search of a tactic-based theorem prover.

For a general, informal introduction to proof planning see Bundy’s “A Science of reasoning” [6]. The use of explicit plans to guide proofs can be found in the proceedings of CADE-9 [4]. An earlier piece of work that led to the development of proof planning was the use of meta-level inference to guide equation solving, implemented in the Press system [33]. This project hopes to add to this field by providing a way to automate the formation of tactics, which would reduce the need for human intervention and hopefully lead the way toward removing one of the labour intensive steps needed in the formation of proof plans. It is hoped that, if successful, this project would allow the automatic formation of tactics using information gathered from existing proof corpuses.

In order to do this one or two suitable proof corpus(es) must be chosen and transformed into a suitable format for the data-mining.

We intend to adapt probabilistic reasoning techniques, such as Variable Length Markov Models (VLMM) [13] and Log Linear Models (LLM) [2], to the iden-

tification of rule sequences that are useful for predicting the next rule. These techniques will be used to discover patterns existing in the proof corpuses. These patterns can be viewed as simple probabilistic tactics.

These simple tactics will be adapted using Koza-style genetic programming [19]. Using this, we will generalise these simple tactics into more complex ones, e.g. containing repetition and conditional branching. This will require the development of an evaluation function for scoring the evolving tactics.

These techniques will be applied to the proof corpuses to allow us to extract new tactics.

The new tactics will be evaluated, e.g. by applying them to a set of test theorems and comparing their performance to the available alternatives.

## 2. Review of Existing Literature

This project depends on the technique of proof planning and on tactics, which were invented by the Mathematical Reasoning Group (MRG) from Edinburgh University. This technique has already been used for a number of systems including  $\lambda$ Clam and  $\Omega$ mega.

### 2.1 Proof Planning Background

A *proof tactic* is a computer program for applying the rules of inference of a mathematical theory library [14]. Tactics are widely used in interactive proof systems for automating common patterns of proof and, hence, improving productivity. Tactic-based theorem provers have been developed both in academia (COQ, HOL, Isabelle, Nuprl, PVS) and industry (Forte, ProofPower). Until recently, this has required the manual construction of tactics. If successful, this project would reduce this impediment.

Proof planning is an approach to theorem proving that uses so-called proof methods rather than low level logical inference rules to find a proof of a theorem at hand. A proof method specifies and encodes a general reasoning strategy that can be used in a proof, and typically expands to a number of individual inference rules. For example, an induction strategy can be encoded as a proof method. Proof planners search for a proof plan of a theorem which consists of applications of several methods. An object level logical proof may be generated from a successful proof plan. Proof planning is a powerful technique because it often dramatically reduces the search space, since the search is done on the level of abstract methods rather than on the level of several inference rules that make up a method. The advantage is that search with methods can be much better structured according to the particular requirements of mathematical domains.

Proof planning also allows reuse of the same proof methods for different proofs, and, moreover, generates proofs where the reasoning patterns of proofs are transparent. When methods are designed appropriately, the level of proof plans can capture the level of communication of proofs amongst mathematicians. Hence proof plans can offer an intuitive appeal to a human mathematician.

Proof planning extends tactic-based theorem proving by augmenting tactics with specifications of their behaviour. These specifications can be used to compose tactics automatically into strategies for complete proofs. A *proof method* is the computational representation of a common pattern of proof in a family of related proofs [6]. A *proof critic* similarly represents a common technique for patching

an initially failed proof attempt [15]. Both methods and critics consist of a tactic plus a specification of its preconditions and effects, expressed in a meta-language describing the syntactic properties of the formulae input to and output by the tactic. Tactics are combined with tacticals to produce higher-level tactics. *Tacticals* include operations of sequencing, non-determinism and repetition. *Proof planning* is the use of AI plan formation technology to guide proof search by constraining it to a set of proof methods and critics. Proof planning limits the combinatorial explosion of potential proof steps which occurs if exhaustive search methods are used.

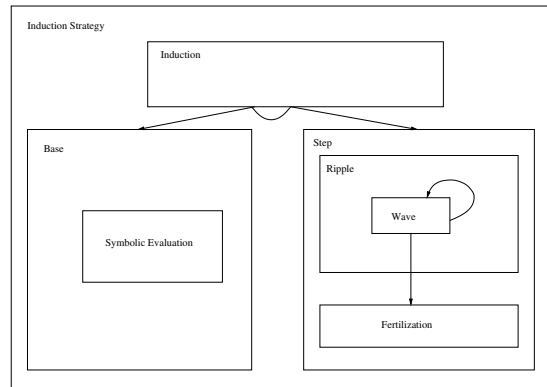


Figure 2.1: A Proof Plan for Inductive Proof

### 2.1.1 $\lambda$ Clam and $\Omega$ mega

The Mathematical Reasoning Group implemented the technique of proof planning in the *Clam* and  $\lambda$ *Clam* proof planners [5] [25] and applied it particularly to the kind of inductive proofs that arise in verification and synthesis of IT systems (see figure 2.1). It has extended the range of problems that can be solved without human intervention. In particular, the use of proof critics has automated the discovery of intermediate lemmas and generalisations [15] - so called, “eureka” steps, which were previously thought to require human intervention.

The  $\Omega$ mega group, based in the Saarland University, Saarbrücken and the German Research Center for Artificial Intelligence (DFKI) developed the  $\Omega$ mega system [1].  $\Omega$ mega is a system with the ultimate purpose of supporting theorem proving in main-stream mathematics and mathematics education. The current system consists of a proof planner and an integrated collection of tools for formulating problems, proving subproblems, and proof presentation.

## 2.2 Previous Learning Work

There have been several previous attempts to learn new proof methods or tactics from example proofs. Also of interest to us are systems which learn and predict patterns, this has been particularly common in the bioinformatics community.

### 2.2.1 Pre-condition Analysis

In his PhD project with the MRG, Bernard Silver applied techniques of explanation-based learning to the automated learning of proof methods for equation solving [31]. His Learning-Press system analysed successful solutions to equations and generalised these solutions to form methods for guiding the Press equation solving system. In this way, he was able to automatically rediscover simplified versions of many of the previously hand-coded methods of Press.

Similarly, another MRG PhD, Roberto Desimone, automated the reconstruction of inductive proof plans, [9]. The techniques of both Silver and Desimone generalised from single successful proofs and required the system to be primed with some key meta-level concepts for expressing the preconditions and effects of the methods they learnt. Silver and Desimone used precondition analysis which learns new inference methods by evaluating the pre- and post-conditions of each inference step used in the proof. A dependency chart between these pre- and post-conditions is created, and constitutes the pre- and post-conditions of the newly learnt inference systems. These methods are syntactically complete proof steps.

### 2.2.2 Learning using Markov Models

Ron, Singer and Tishby created a distribution learning algorithm for Variable memory Length Markov Models [27]. These processes can be described as a subclass of probabilistic finite automata (PFA) which they call Probabilistic Suffix Automata (PSA). Though hardness results are known for learning distributions generated by general probabilistic automata, they prove that the algorithm they present can efficiently learn distributions generated by PSAs. In particular, they show that, for any target PSA, the KL (Kullback-Liebler)-divergence between the distribution generated by the target and the distribution generated by the hypothesis the learning algorithm outputs, can be made small with high confidence in polynomial time and sample complexity. The learning algorithm is motivated by applications in human-machine interaction. In their paper, they present two applications of the algorithm. In the first one they apply the algorithm in order to construct a model of the English language, and use that model

to correct corrupted text. In the second application they construct a simple stochastic model for *E.coli* DNA. As with our project, they looked at data which has a *short memory property*, i.e. consider the empirical probability distribution on the next symbol in a sequence given the preceding symbols, then there exists a length  $L$  (*memory length*) such that the conditional probability distribution does not change substantially if we condition on preceding subsequences of length greater than  $L$ . These can form Markov models of order  $L > 1$ , they give efficient procedures both for generating sequences and for computing their probabilities.

Markov Models have been frequently used in Bioinformatics, especially for classifying incomplete DNA strands. Some of the work on pattern matching in DNA sequences [3], as in the GENOME project, is related to our learning mechanism.

### 2.2.3 Learning Proof Methods

More recently, Kerber, Jamnik, Pollet and Benzmüller, from Birmingham University, have applied the techniques of least general generalisation to a family of similar proofs to learn new proof methods for various domains [16]. They present a framework for automated learning within mathematical reasoning systems. In particular, this framework enables proof planning systems to automatically learn new proof methods from well chosen examples of proofs that use a similar reasoning pattern to prove related theorems. Their framework consists of a representation formalism for methods and a machine learning technique which can learn methods using this representation formalism. They present an implementation of this framework, called Learn $\Omega$ Matic, which adds new methods to the  $\Omega$ mega proof planner. Methods are represented using a regular grammar over individual proof steps and previously learned methods, allowing a hierarchical collection of methods. Note that this technique requires all the proofs in the family to be examples of the learned method.

### 2.2.4 Random Fields

Stephen Della Pietra, Vincent Della Pietra and John Lafferty presented a technique for constructing random fields from a set of training examples in their paper *Inducing Features of Random Fields* [7]. Their learning paradigm builds increasingly complex fields by allowing potential functions, or features, that are supported by increasingly large subgraphs. Each feature has a weight that is trained by minimizing the KL-divergence between the model and the empirical distribution of the training data. A greedy algorithm determines how features are incrementally added to the field and an iterative scaling algorithm is used to estimate the optimal values of the weights. The random field models and techniques

introduced in this paper differ from those common to much of the computer vision literature in that the underlying random fields are non-Markovian and have a large number of parameters that must be estimated. Relations to other learning approaches, including decision trees, are given. As a demonstration of the method, they described its application to the problem of automatic word classification in natural language processing.

### 2.2.5 Learning Heuristic Control

Sculz 2001 [30], which is a continuation of previous work such as [11, 8], investigates learning of heuristic control knowledge in the context of machine oriented theorem proving, more precisely, equational or superposition based theorem proving. Knowledge gained from the analysis of the inference process is used to learn important search decisions, which are represented as abstract clause patterns. These are employed in heuristic evaluation functions to better guide the search when attacking new proof problems. The selection of heuristic evaluation functions for a new problem at hand is guided by meta-data. Unlike our project, the learnt information in Schulz's work is not becoming a reasoning primitive, such as our learnt methods. It rather guides the search amongst the existing primitives at the global search layer instead of building up new, structured chunks of encapsulated search processes.

### 2.2.6 Proof Reuse and Simulation of Human Learning

Kolbe, Walter, Brauburger, Melis and Whittle have done related work on the use of analogy [21] and proof reuse [18, 17]. Their systems require a lot of reasoning with one example to reconstruct the features which can then be used to prove a new example. The reconstruction effort needs to be spent in every new example for which the old proof is to be reused. In contrast, we learn our reasoning patterns from a large number of examples.

A piece of related work in Cognitive Science is Furse's Mathematics Understander [12], MU, which stores mathematical domain and procedural knowledge in a contextual memory system, and tries to simulate how students learn mathematics from textbooks. MU builds up a uniform low-level data structure, while we build high-level hierarchical proof planning methods.

In terms of a learning mechanism, more recent work on learning regular expressions, grammar inference and sequence learning [34] is related. Learning regular expressions is equivalent to learning finite state automata, which are also recognisers for regular grammars. Muggleton has done related work on grammatical inference methods [23] which automatically constructs finite-state structures from

trace information. His method IM1 is a general one and can describe all other existing grammatical inference methods. IM1 consists of first, generating a prefix tree from example traces, second, merging of states to get canonical acceptor states (which still describe only the example traces), and third, merging states which essentially does the generalisation of the structure. The generalisation, i.e., merging, is determined by a particular chosen heuristic measure. The existing state automata learning techniques differ depending on the heuristic that they employ for generalisation. These techniques typically require a large number of examples in order to make a reliable generalisation, or supervision or an oracle which confirms when new examples are representative of the inferred generalisation. Furthermore, the techniques employed by Muggleton learn only sequences, i.e. regular expressions whereas ours will most likely include constant repetitions of expression and expressions represented as trees. This is particularly relevant when we must choose a grammar to represent our tactic language.

There have been various approaches to incorporate learning in planning. In the PRODIGY system [35] a number of techniques for learning are available. The goal of the learning process is either to get control knowledge, that is, rules that describe which goal to tackle next and which method to prefer at the decision points of the planning algorithm, or learn planning operators from the change of planning states by observing an expert agent. Our plan differs in both aspects as our goal is to learn new operators that are learnt from other operators and could be compared to learning of macro operators or chunks [28]. Another difference is that learning from an analysis of the domain theory without the generation of examples appears to be difficult, since proof planning methods are complex and the post-conditions are only available when a method is applied in a concrete proof situation.

## 2.3 Previous Work on Genetic Programming

In order to evolve the naive patterns to obtain new tactics, we have decided to use Genetic Programming (GP). GP will allow us to evolve the tactic(s) without insisting that we know exactly which will be best. The nature of GP is that we can find several solutions and then check which is most appropriate.

### 2.3.1 Koza

John Koza explains the principals of Genetic Programming in his book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [19]. Koza's work describes and illustrates genetic programming with 81 examples

from various fields, of particular interest is the ‘Evolution of Subsumption’, which will be similar to the strategy we wish to use.

Koza’s approach genetically breeds populations of computer programs to solve problems by executing three steps:

1. Generate an initial population of random compositions of the functions and terminals.
2. Iteratively perform the following sub-steps until the termination criterion has been reached:
  - (a) Execute each program in the population and assign it a fitness value
  - (b) Create a new population by:
    - (i) Reproduction: Copy existing programs to the new population
    - (ii) Crossover: Create two new programs by genetically recombining randomly chosen parts of two existing programs
3. The best program at the time of termination is deemed to be the result of the genetic programming. This may be a complete or partial solution.

Although Koza describes his technique in terms of programs, functions and terminals, it is easy to see how this could apply to tactics, proof steps and operations from our grammar.

### 2.3.2 Evolutionary Programming

John Levine and David Humphreys’ [20] developed L2Plan (learn to plan), a genetic programming based method for planning. Their system represents control knowledge as a *policy* and learns using Genetic Programming. The program’s crossover and mutation operators are augmented by simple local search. L2Plan was able to produce policies which solved all the test problems it was given, outperforming hand-coded policies written by the authors. The genetic programming used for this would be well suited to our task, randomly generating an initial population and then evaluating their fitness against our test set may well produce results that would be difficult to find using other methods.



### 3. Description of Central Ideas

Automatic learning by reasoning systems is a difficult and ambitious problem. Our work demonstrates one way of starting to address this problem, and by doing so, it presents several contributions to the field.

1. Although machine learning techniques have been around for a while, they have been relatively little used in reasoning systems. Making a reasoning system learn proof patterns from examples, much like students learn to solve problems from examples demonstrated to them by the teacher, is hard. Our work makes an important step in a specialised domain towards a proof planning system that can reason *and* learn.
2. Proof methods have complex structures, and are, hence, very hard to learn by the existing machine learning techniques. We approach this problem by abstracting only as much information from the proof method representation as needed, so that the machine learning techniques can handle the information. Later, after the reasoning pattern is learnt, the abstracted information will be restored to its original form as much as possible.
3. Unlike in some of the existing related work, we are not aiming to improve ways of directing proof search within a fixed set of primitives. Rather we aim to learn the primitives themselves, and to investigate whether this improves the framework and reduces the search space within the proof planning environment. Instead of searching amongst numerous low level proof methods, a proof planner can search with a newly learnt proof method which encapsulates several of these low level primitive methods.

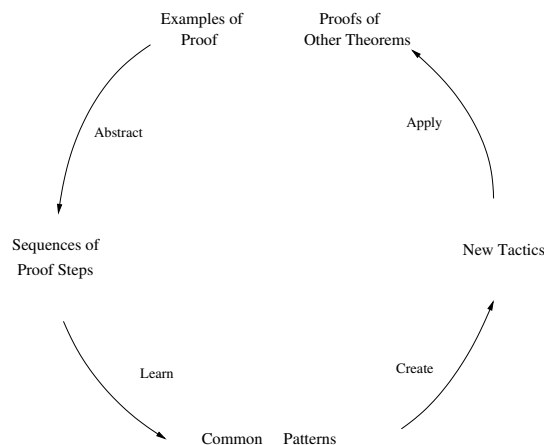


Figure 3.1: The structure of our approach to learning proof methods.

### 3.1 Choosing the Corpus

A suitable corpus of proofs to be used in this project must be chosen, this corpus must meet precise requirements.

1. It must be stored in computational form, so that it is available for Data-Mining
2. It must be sufficiently large to contain many examples of multiply occurring patterns of proof
3. There must be an appropriate diversity of kinds of proof steps, i.e. sufficient different kinds of proof steps that patterns can be identified, but not so much diversity that patterns do not recur. Note that the appropriateness of diversity is relative to corpus size: the larger the diversity, the larger the corpus required for the re-occurrence of patterns.

Note first that the huge search space generated by resolution-style theorem provers are, unfortunately, mostly unsuitable because of requirement 3 above: typically only one or two rule of inference are used. We could try to differentiate rule applications by the formulae they manipulate, but these formulae are generated during the proof search and are too diverse, e.g. millions of derived clauses. In addition, it has been suggested that interactive theorem provers may be more likely to yield interesting patterns due to the structure that people insert in their proofs. Conversely, it has also been suggested that a wholly automatic theorem prover may yield patterns as it searches for proofs in a formulaic way.

#### 3.1.1 Isabelle as a Candidate

Isabelle is an interactive theorem prover developed at Cambridge [24]. It satisfies the necessary criteria:

1. Isabelle's theory libraries are available online, and they also come with the implementation (which is also available online).
2. Isabelle has several hundred theory files, including a large number based on its Higher Order Logic (HOL).
3. Isabelle HOL has a relatively small basis of theorems and Higher Order Logic axioms, all subsequent theorems are built upon these (and upon earlier theorems). It is possible to deconstruct any theorem to this basis, or to any lower level which provides the appropriate diversity.

Isabelle has some inbuilt commands which allow the proof of a theorem to be extracted.

### 3.1.2 Mizar as a Candidate

Mizar was developed at the University of Bialystock in Poland as an aid to the development of mathematical articles for formalized maths [29]. Mizar also meets the necessary requirements:

1. Mizar's theory libraries are stored electronically, they are available online, and via the implementation.
2. The Mizar Mathematical Library contains the proofs of several thousand major mathematical theorems.
3. The proof steps we are interested in are produced by Mizar's inbuilt verifier, this has a relatively small number of possible steps.

Extracting the corpus in the form we require is slightly more complex with Mizar, this is because of the inbuilt verifier, which is the part of Mizar which checks the validity of the proof and produces the proof steps we require.

A Mizar article may have the following form:

```

environ
environment section
begin
theorem statement
proof
a1: statement a1;
a2: statement a2;
...
b1: statement b1 by a1,a2 and THEORY:Theorem;
end;
```

The environment section at the beginning tells the Mizar system where the information used in the article can be found. Any notation must be contained in one of the files named in the notation section, any theories used must come from a theory file named in the theory section etc. Note that when a theory is used, the theory file name and the name of the theorem within that file must be used as an identifier (*THEORY:Theorem*). The verifier would take statement b2 and attempt to prove its validity using statements a1, a2 and the theorem named. It uses simple rewrite rules to check that this is a valid proof step. It is this information from the verifier that we are interested in.

## 3.2 Data-Mining the Proof Corpuses

Once the proof corpuses have been put into a suitable format, probabilistic reasoning techniques will be used to identify patterns within the proof structures, these patterns will form the basis for the new tactics.

Variable Length Markov Models (VLMMs) seem to offer the ideal solution to the task of identifying patterns. Using the proof corpus, a VLMM will be trained and then used to predict the next proof step. The main advantage of using VLMMs is the variable length. Each step is predicted using the likelihood of it appearing given the string that has appeared before it. One problem of using other techniques is that longer strings would be prejudiced against due to the fact that they are less likely to appear simply by chance. For example, a ‘pattern’ of rule A followed by rule B would possibly occur (say)  $n$  times simply by chance, but a pattern of ABCDCBC should be considered to be significant if it happened to occur  $n$  times. The very nature of VLMMs means that this problem would be simply dealt with.

Although there is existing software which deals with pattern formation and Variable Length Markov Models [26, 10] the software was very specifically for DNA pattern construction and some experiments and investigation proved that adapting this software would be prohibitively complex.

One of the major problems encountered with the pattern discovery so far is the case splits in the proofs. Although many pieces of software exist for identifying patterns in strings (most commonly for DNA sequences, but also for more general strings) which could be adapted for use with proof structures, we have been unable to identify any existing software which identifies patterns within tree structures. It was suggested that case splits could be ignored or simply treated as a special case i.e. a “split token”, however, the frequency of occurrence of some sort of branching structure within a proof means that in this case we may well lose many interesting patterns.

The technique decided upon was to split the proofs into separate strings and give weights accordingly i.e. all the steps at the end of any branch have weight 1, before each split the weights are given as  $1/\text{branches} * \text{weight after split}$  – so a tree which has 3 two-way splits would have a weight of 1 at the end of each branch, 0.5 on every branch between the last two splits, 0.25 after the first split and 0.125 before there is ever a split point. For example, a split point such as that shown in figure 3.2 is now represented in a list of tuples form:

$$[[[0.5, A], [0.5, B], [0.5, C], [1, D], [1, F]], [[0.5, A], [0.5, B], [0.5, C], [1, E], [1, G]]] \quad (3.1)$$

This is better shown by figure 3.3. These weights are incorporated simply at the point where the Markov Model is updated. It would be much more elegant to

have software which learned markov models directly from the tree structures but this has not yet been found.



Figure 3.2: Example proof steps with split

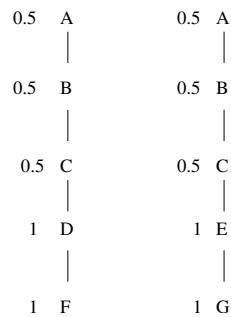


Figure 3.3: Proof steps after weighting

### 3.3 Genetic Programming

The simple tactics constructed by the probabilistic methods described previously will only consist of fixed length combinations of particular rules. This does not reflect the full generality of hand-built tactics. For instance, recursion might be used to capture the repeated application of a particular sub-tactic a variable number of times. Or conditionals might be used to capture variations in the particular rule combinations. To generalise our initial tactics in this way, we turn to genetic programming.

Genetic programming evolves a computer program to meet a specification of its desired behaviour. It works by genetically breeding a population of computer programs, using the principles of Darwinian natural selection and biologically inspired operations. We have identified the genetic programming techniques of Koza as being well suited to evolving compound object such as tactics [19].

Genetic programming usually starts by creating some random programs and then running each candidate program on a set of fitness cases: the more cases for which the program produces the correct result, the higher the fitness. The fittest

programs are selected for evolution and the procedure recurses until a program exceeds a predetermined fitness level. We will adapt this technique to our application. The Markov models, output from the probabilistic reasoning stage, will be used both as a subset of the initial programs and as the fitness cases, i.e. we will treat the Markov models as the starting tactic and measure the fitness of each evolved tactic partially by how many of these Markov models it subsumes.

Some care is required over the design of the tactic language. The choice ranges from the Birmingham-style regular grammars, via a limited set of tacticals to a general programming language, such as ML. A parsimonious language will be better suited to genetic programming, e.g. a limited set of tacticals. Moreover, the language must not require information that cannot be obtained by analysis of the proof corpus. For instance, it is no use including while-loops or if-then-else, if their conditions cannot be identified. Non-conditional forms of repetition and non-determinism must be used instead.

## 4. Work so Far

The project is currently proceeding as expected, with a number of objectives having been completed.

### 4.1 Extracting the Data

Work has been performed with the theorem prover Isabelle to extract and format the proof scripts. The theorem prover Mizar has a huge library of mathematical theorems, this is being used in the same way as Isabelle to allow a comparison between different provers. Two programs have been tested as a basis for pattern discovery: Sparse Markov Transducers (SMT) from Eleazar Eskin at the University of Columbia [10] and the Teiresias algorithm from the IBM research group [26].

Extraction of information from Isabelle was eased significantly by some inbuilt tools which allow the extraction of a proof script via the theorem name. Some work was required to allow many theorem proofs to be extracted at once and also to transcribe this information into a suitable format.

The information taken from Isabelle has been used for testing with a variety of software. Although this software proved unsuitable for our purposes, it has still been possible to use this software to provide a number of results. A detailed comparison of the above two pieces of software has been carried out.

Analysis of available software and problems with it has led to the design and production of some new software with the specific purpose of finding patterns within tree structures.

### 4.2 Data-Mining Software

The SMT algorithm works by forming a tree based on the Markov Models given by training data which in our case are the proofs described above. This prediction tree has been used to provide the likelihood of certain strings appearing together as patterns.

The results from the SMT algorithm show a number of patterns. However, it is noticeable that all the patterns begin the same and also are significantly less varied than those given by the Teiresias algorithm. This is due to the way that

the SMT algorithm works. The SMT program was developed specifically for completing amino acid sequences. This means that the training data (which is the proofs in our case) is assumed to be always in a specific position. For our purposes the position that the tactic names appear within the proof is unimportant with only the positions with respect to other tactics being noticed. For this reason, some of the patterns may in fact not be patterns but a trait of a certain tactic appearing at a certain point in the proof. These problems were not all foreseen before testing was carried out, but the difference in intention was seen to be a problem and so another algorithm was sought.

The Teiresias algorithm does not make use of Variable Length Markov Models as intended, instead it finds patterns using a scanning algorithm which counts the occurrences of each pattern. A user-defined parameter allows the number of occurrences required to define a pattern to be specified. The program then combines the patterns allowing ‘wild-card’ characters to find more general sequences. The algorithm returns the most specific sequences which still have the same number of occurrences. Even with patterns which must occur at least 20 times, a significant (and varied) number of patterns are found. When only 10 occurrences are required to define a pattern, over 440 are found in just a few seconds.

It appears that, of the two, despite the fact that the SMT algorithm works using VLMM as we desired, the Teiresias algorithm is far better suited to our purposes. However, this algorithm also has significant drawbacks;

1. It does not notice the significance of two tactics always occurring together if they only appear a small number of times. For example, if a step *A* only occurs 19 times, but 18 of these are followed by step *B*, the Teiresias algorithm would not notice this to be a pattern.
2. There is no significance given to longer patterns. For our purposes it would be desirable for long patterns to have to occur fewer times to constitute a pattern, however, the Teiresias algorithm treats all strings the same no matter what the length.
3. It does not allow tree structures to be used as an input, nor is there any intention for this to be developed. This is obviously a major concern for our project.

### 4.3 Design Specific Software

The problems found when analysing these two pieces of software appeared to indicate the problems of using any existing software. No software that better suited

our needs has been found, so a specific piece of software has been designed from scratch. This software uses the ‘weighting’ technique described in the previous chapter and has been successful so far.

The software takes as input trees represented in a list-of-lists formation. These trees are then linearised using the ‘weighting’ technique. The program also extracts the branching points, the proof step names, and the number of occurrences of each proof step before training the Markov Models.

Any proof step which results in a branch is taken note of along with the first node on the branch as shown by figure 4.1

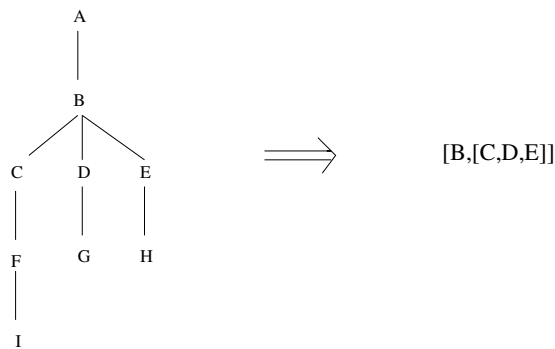


Figure 4.1: Example of the Branching Information Recorded

This branching information will be used during the genetic programming stage in order to reconstruct the branches where possible.

For each point in the proof a number representing the likelihood of that step occurring given the previous steps is calculated. This is calculated by multiplying the number of times that step occurs in the corpus by the weight given to the particular sequence. This means that the sequence  $A - B - C - D$  is calculated in terms of  $D$ .  $P(D|A - B - C) = O(D) * W$  (where  $O(D)$  is the number of times  $D$  occurs). In such a string  $P(D|A)$  and  $P(D|A - B)$  would also be calculated. If any of these strings have occurred before, (for example if  $P(D|A - B - C)$  is already in the database), then the calculated probability is added to the existing probability.

The nature of this software is such that a lot of redundancy is created - this must be to ensure that sub-patterns that occur more frequently than the longer patterns have their probability updated correctly.

Finally, any patterns which have a probability above the specified threshold are returned. Although any patterns which are directly subsumed by others with the same probability are deleted. These are created as above and as the probabilities are the same, we are interested in the longer one.

## 4.4 Feasibility Tests

One of the concerns about the project (which was also brought up in the queries section) was that even using human-directed proofs, a slight change in the order of steps being applied may well lead to a pattern being missed. It was decided that it would be worthwhile to assign each of the theorems of Isabelle's HOL library to a class and then look for patterns of classes. This approach seemed likely to give some useful results because of techniques (or perhaps habits) used by both people and automatic provers when looking for a proof. It was suggested that there is a tendency to perform steps from the same class together. i.e. it is often the case that people begin by using all possible rewrite rule to simplify the goal as much as possible. It is not inconceivable that many theorem provers also use a similar principal for heuristics.

A hand comparison of a number of similar proofs has been carried out. The intention of this was to examine if the similarities in the theorem translated into a similarity in the proof. For the most part, it was found that this was the case, however, the examples studied were simple and it is very possible that two complex theorems proven by two different people may well give very different proofs, even if the statement of the theorems are almost identical.

A period of two months were spent in Poland at the University of Bialystock learning the Mizar language and evaluating its suitability. During this period a Mizar article was written as part of the learning process. Work is ongoing on transcribing the Mizar Mathematical Library into a suitable format.

# 5. Future Targets

## 5.1 Milestones

1. Complete Transformation of Mizar Theories [Ongoing]

Transforming the Mizar Mathematical Library into a suitable format is much more complicated than transforming the Isabelle library due to the Mizar verifies. However, work on this stage is well underway and will continue as and when more information is needed for testing.

2. Find Patterns Using Software [Ongoing]

Although the software is complete and has been tested and the initial patterns extracted from the corpuses, at any point during the project we may require to reuse the developed software to extend our data by using more of the available proof corpuses or by examining different kinds of mathematical theories or logics.

3. Decide on a Tactic Grammar [Oct '03 - Dec '03]

Before any Genetic Programming can be used to evolve the tactics, a grammar that can express these appropriately for our needs must be decided upon.

4. Convert into Tactics using Genetic Programming [Dec '03 - Jul '04]

The principal computational step remaining will be to convert the patterns to tactics using genetic programming. The tactics will have to be formed using some sort of grammar, possibly a regular grammar similar to that used by Jamnik et al. The patterns discovered in step 2 will be combined using genetic programming to give something which resembles a proof tactic much more closely. This step cannot begin until the previous two steps have been completed.

5. Analysis of Results [Jul '04 - Dec'04]

The tactics formed by the previous section must be tested and the results analysed. We hope to be able to reconstruct some existing theorems which are frequently used as well as to re-form some proofs in such a way that they resemble the original proofs in the training data. Another use and as such test of the results

would be to use our project as a way to suggest a next step in an interactive theorem prover (e.g. Isabelle). It would be helpful to test our project by examining how often a useful next step is suggested. This would also require a way to determine a measure of ‘usefulness’. It would also be worthwhile to use an automatic theorem prover which uses proof planning (such as  $\Omega$ mega) and testing whether using our learned tactics as heuristics improves the time taken to find proofs in general.

6. Thesis Write-up [Dec '04 - Jun '05]

## 5.2 Dependencies

Although work up to this point has been able to be carried out concurrently (for the most part) the future stages have quite strict dependencies (these are also shown in figure 5.1).

- Both developing the pattern discovery software and transcribing the Mizar library can be tackled at the same time
- Finding the patterns cannot be performed until the pattern discovery software is completed. Although this can be begun using only the information gained from the Isabelle library, it would be better to have the transformation of the Mizar libraries complete before this stage so that the results can be compared continuously.
- The Genetic Programming stage cannot be begun until at least some results are available from the pattern finding stage. Again, although this can be begun beforehand, it would be better to have as much information as possible before beginning this stage
- Obviously, the analysis of the results stage cannot be properly carried out until the results are available. However, there are a number of intermediate analyses that can be used to indicate the success at certain stages
- The thesis will be ongoing from a fairly early stage but again the majority of this stage of the project will have to wait until the previous stages are complete.

There are a few milestones which will indicate the progress of the project:

- Find known existing pattern

It would be a significant boost to ‘rediscover’ a pattern which we already know exists. For example, if a (commonly used) theorem

is deconstructed into its proof script, we would expect to find this as a pattern. It should be possible to reach this milestone after finding the patterns, it is not necessary to wait until after the genetic programming step.

- Find ‘intuitive’ patterns

It is hoped that we will find some patterns which intuitively make sense. Although this is aimed for after the genetic programming step, it should be possible to see the potential for these after the pattern finding step. One example is that people tend to begin by eliminating quantifiers in mechanical proofs, we would hope to form a tactic which contains a repetition of quantifier elimination steps at the beginning. This is suggested by Dale Miller’s work on Normal Proofs [22].

- Re-form a proof

Once the genetic programming stage has been completed and testing of out new tactics begins, we can hope that we will be able to re-form a proof of one of the theorems from our training data. We would expect that we would find some proofs which are developed in such a way as to be very similar to the originals from our training data.

## 5.3 Deliverables

- Software

The fully working software to be used on our corpuses. This is currently ready – although some minor changes may be made later if more information is required at the genetic programming stage.

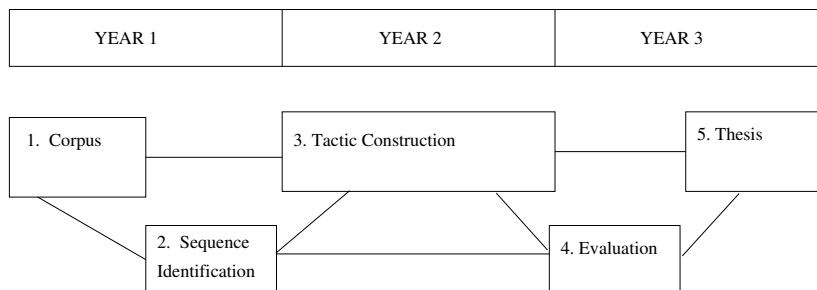


Figure 5.1: A Timeline Showing Task Dependencies

- Paper on pattern discovery

It has been suggested that the lack of any readily-available software for pattern discovery with trees suggests that it may be useful to write a paper about the problems encountered and the resulting software.

- Genetic Programming Software

Work has begun on the software to perform the genetic programming step and evolve the patterns into tactics.

- Tactics

Once all the software has been developed and tested to find the most useful specifications, we will need to produce patterns from a large data-set and evolve these into tactics to be used for analysis.

- Test Data Results

The results of testing the final tactics on various systems and in various ways. e.g. The tactics could be tested as a heuristic in an automatic proof planner such as  $\Omega$ mega and it could be tested as a proof assistant (suggesting future steps) in an interactive proof planner such as Isabelle.

- PhD Thesis

## 6. Likely Outcome

As stated, the intention of the project is to allow the automatic formation of tactics to be used in proof planning and automatic theorem proving. It is hoped that this project will allow a way to provide tactics which will help guide proof search and will reduce the amount of human intervention needed for proof planning. If successful, this procedure should be able to be integrated into automatic theorem provers and should help reduce the search necessary to find a successful proof. This, in turn, should help improve the success rate of the theorem prover. Theorem provers generally have a maximum time for finding a successful proof so improving the search direction should allow an increase in the number of theorems which can be proved before a timeout occurs.

On a much more localized view, the project would be viewed to be a success if we could demonstrate that significant patterns had been found and that sensible tactics had been formed using these patterns. It would be hoped that tactics could be discovered which made a difference in the search space required to find proofs of a certain type – or even the likelihood of certain types of proof succeeding. Some principles can already be noticed using intuition and common sense, such as the principle that rewrite rules generally occur within a cluster of rewrite rules, and that many proofs begin with the elimination of quantifiers. It would seem a reasonable hope that tactics representing these (or similar) observations could be found.

There are many ways to check the successfulness of the project at various stages. At the final stage it would seem appropriate to evaluate the produced tactics by inspecting them for mathematical “sensitivity”, i.e. do they make sense within the context of the theorems studied? Do they seem like a sensible approach? A more concrete evaluation would be to enter these new tactics as heuristics within an automatic theorem prover and look for any changes/improvements in its performance.

At a much earlier stage, one suggested evaluation is to see if the patterns found show any expected results, such as existing lemmas. This would be expected to occur when using the Isabelle library (for example) as theorems are used as proof steps, if each of these theorems are broken up into their proof traces and so on until the low level HOL logic theorems are reached, then we would expect the proof traces of commonly used lemmas to show up as frequently occurring patterns.



# Bibliography

- [1] C. Benz Müller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, K. Kohlhase, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. *Omega: Towards a mathematical assistant*. In W. McCune, editor, *14th International Conference on Automated Deduction*, pages 252–255. Springer-Verlag, 1997.
- [2] A. Berger. The improved iterative scaling algorithm: A gentle introduction, 1997.
- [3] A. Brazma and K. Cerans. Efficient learning of regular expressions from good examples. In S. Arikawa and K. P. Jantke, editors, *Algorithmic Learning Theory: Proc. of the 4th International Workshop on Analogical and Inductive Inference, AII'94*, pages 76–90, Berlin, Heidelberg, 1994. Springer.
- [4] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.
- [5] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.
- [6] Alan Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.
- [7] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing feature of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.
- [8] J. Denzinger and S. Schulz. Learning Domain Knowledge to Improve Theorem Proving. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th CADE, New Brunswick*, number 1104 in LNAI, pages 62–76. Springer, 1996.
- [9] R. V. Desimone. Learning control knowledge within an explanation-based learning framework. In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning – Proceedings of 2nd European Working Session on Learning, EWSL-87, Bled, Yugoslavia*. Sigma Press, May 1987. Also available from Edinburgh as DAI Research Paper 321.

- [10] Eleazar Eskin, William Noble Grundy, and Yoram Singer. Protein family classification using sparse markov transducers. *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, August 20-23, 2000.
- [11] M. Fuchs and M. Fuchs. Feature-based learning of search-guiding heuristics for theorem proving. *AI communications*, 11:175–189, 1998.
- [12] E. Furse. Learning university mathematics. In C. Mellish, editor, *Proceedings of the 14th IJCAI. Vol. 2. International Joint Conference on Artificial Intelligence*, pages 2057–2058. Morgan Kaufmann, 1995.
- [13] Aphrodite Galata, Neil Johnson, and David Hogg. Learning variable length markov models of behaviour. *Computer Vision and Image Understanding: CVIU*, 81(3):398–413, 2001.
- [14] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
- [15] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available from Edinburgh as DAI Research Paper No 716.
- [16] M. Jamnik, M. Kerber, and M. Pollet. Automatic learning in proof planning. In F. van Harmelen, editor, *Proceedings of 15th ECAI. European Conference on Artificial Intelligence*, 2002. Forthcoming.
- [17] T. Kolbe and J. Brauburger. Plagiator – a learning prover. In W. McCune, editor, *Lecture Notes in Artificial Intelligence, No. 1249*, pages 538–552. Springer Verlag, 1997.
- [18] Thomas Kolbe and Christoph Walther. Proof analysis, generalization and reuse. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications, Vol. II Systems and Implementation Techniques*, Applied Logic Series, vol. 9, pages 189–219. Kluwer Academic Publishers, Dordrecht, Boston, London, 1998.
- [19] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.
- [20] John Levine and David Humphreys. Learning action strategies for planning domains using genetic programming.
- [21] E. Melis and J. Whittle. Analogy in inductive theorem proving. *Journal of Automated Reasoning*, 22(2), 1998.
- [22] D. Miller. *Proofs in Higher Order Logic*. PhD thesis, Carnegie Mellon University, 1983.

- [23] S. H. Muggleton. *Inductive Acquisition of Expert Knowledge*. Addison Wesley, Reading, MA, 1990.
- [24] L.C. Paulson. *Isabelle: A generic theorem prover*. Springer-Verlag, 1994.
- [25] J. D. C Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with Lambda-Clam. In Claude Kirchner and H el ene Kirchner, editors, *15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 129–133, Lindau, Germany, July 1998.
- [26] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *Bioninformatics*, January(14(1)), 1998.
- [27] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25, 1996.
- [28] P. Rosenbloom, J. Laird, and A. Newell. *The Soar Papers: Readings on Integrated Intelligence*. MIT Press, 1993.
- [29] P. Rudnicki. An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*, Bastad, 1992. Chalmers University of Technology. See <http://mizar.org> for up-to-date information on Mizar and the Journal of Formalized Mathematics.
- [30] S. Schulz. Learning Search Control Knowledge for Equational Theorem Proving. In F. Baader, G. Brewka, and T. Eiter, editors, *Proc. of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, volume 2174 of *LNAI*, pages 320–334. Springer, 2001.
- [31] B. Silver. *Using Meta-Level Inference To Constrain Search And To Learn Strategies In Equation Solving*. PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1984. Published as a book by North Holland.
- [32] H. A. Simon and A. Newell. Heuristic problem solving: The next advance in operations research. *Operations Research*, 6(1), 1958.
- [33] L. Sterling, Alan Bundy, L. Byrd, R. O’Keefe, and B. Silver. Solving symbolic equations with PRESS. In J. Calmet, editor, *Computer Algebra, Lecture Notes in Computer Science No. 144.*, pages 109–116. Springer Verlag, 1982. Also available from Edinburgh as DAI Research Paper 171.
- [34] R. Sun and L. Giles. Sequence learning: Paradigms, algorithms, and applications. In *Lecture Notes in Artificial Intelligence*, number 1828. Springer Verlag, 2000.
- [35] Veloso, Carbonell, Perez, Borrajo, Fink, and Blythe. Integration planning and learning: The prodigy architecture. In *Journal of Experimental and Theoretical Artificial Intelligence 7 (1)*, pages 81–120, 1995.