# Learning Proof Methods in Proof Planning

Mateja Jamnik, Manfred Kerber and Christoph Benzmüller
School of Computer Science, University of Birmingham; Birmingham B15 2TT, England, UK
{M.Jamnik, M.Kerber,C.Benzmuller}@cs.bham.ac.uk

## 1 Introduction

Our research interests in this project are in exploring how automated reasoning systems can learn theorem proving strategies. In particular, we are looking into how a proof planning system (Bundy, 1988) can automatically learn new proof methods. For more information on this work, see (Jamnik et al., 2000, 2001).

One of the ways to extend the power of a proof planning system is to enlarge the set of available proof methods. This is particularly beneficial when a class of theorems can be proved in a similar way, hence a new proof method can encapsulate the general structure, i.e., the reasoning strategy of a proof for such theorems. A difficulty in applying a proof strategy to many domains is that in the current proof planning systems new methods have to be implemented and added by the developer of a system. In this work, our aim is to explore how a system can learn new methods automatically given a number of well chosen examples of related proofs of theorems. This would be a significant improvement, since examples (e.g., in the form of classroom example proofs) exist typically in abundance, while the extraction of methods from these examples can be considered as a major bottleneck of the proof planning methodology.

In our work we devised an approach to automatic learning of proof methods within a proof planning framework. This research bridges at least two well established AI areas: automated reasoning and machine learning. From the automated reasoning point of view our work is interesting, because it aims to improve the reasoning systems by using machine learning techniques. From the machine learning point of view, our work can be seen as an interesting application of machine learning techniques in automated reasoning systems.

Figure 1 gives a structure of our approach to learning proof methods. Given some examples of proofs that use a similar reasoning strategy, we need to represent them so that the examples are amenable to the learning process. This is achieved by abstracting proof representations into sequences of method specifiers. We then use our learning algorithm to learn the so-called method outlines. A method outline is a representation common to all examples. However, this representation is not yet a full-fleshed proof method. We need to enrich it so that the newly learnt methods can be used in a proof planner for proofs of other theorems. We use precondition analysis to acquire the information for extending the method representation. We implemented this framework within the proof planner of ΩMEGA (Benzmüller et al., 1997).

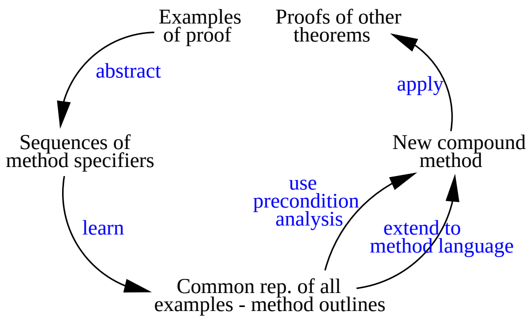
Examples of proof
Proofs of other theorems
abstract
apply
Sequences of method specifiers
New compound method
use precondition analysis
learn
extend to method language
Common rep. of all examples - method outlines

Figure 1: An approach to learning proof methods.

## 2 Example

Consider two examples of proof fragments from group theory. They consist of simplifying an expression using a number of primitive simplification methods such as axioms of identity, inverse and associativity (where $e$ is the identity element, $i$ is the inverse function, and LHS $\Rightarrow$ RHS stands for rewriting LHS to RHS):

$$\begin{aligned} X \circ (Y \circ Z) &\Rightarrow (X \circ Y) \circ Z && \text{(A-l)} \\ e \circ X &\Rightarrow X && \text{(Id-l)} \\ X \circ X^i &\Rightarrow e && \text{(Inv-r)} \\ X^i \circ X &\Rightarrow e && \text{(Inv-l)} \end{aligned}$$

Here are the two examples of proof steps which simplify given expressions:

**Example 1**

$$\begin{array}{l} a \circ ((a^i \circ c) \circ b) \\ \quad \Downarrow \text{(A-l)} \\ (a \circ (a^i \circ c)) \circ b \\ \quad \Downarrow \text{(A-l)} \\ ((a \circ a^i) \circ c) \circ b \\ \quad \Downarrow \text{(Inv-r)} \\ (e \circ c) \circ b \\ \quad \Downarrow \text{(Id-l)} \\ c \circ b \end{array}$$

**Example 2**

$$\begin{array}{l} a^i \circ (a \circ b) \\ \quad \Downarrow \text{(A-l)} \\ (a^i \circ a) \circ b \\ \quad \Downarrow \text{(Inv-l)} \\ e \circ b \\ \quad \Downarrow \text{(Id-l)} \\ b \end{array}$$

The first example can be summarised in the following string of method identifiers: {A-l, A-l, Inv-r, Id-l}. The second example can be summarised in the following string of method identifiers: {A-l, Inv-l, Id-l}. It is clear that the two examples have a similar structure which could be captured in a new simplification method. In pseudo-code, one application of such a simplification method could be described as follows:

**Precondition:** *there are subterms in the initial term that are inverses of each other, and that are not separated by other subterms, but only by brackets.*

**Tactic:** 1. *apply associativity* (A-l) *for as many times as necessary (including* 0 *times) to bring the subterms which are inverses of each other together, and then*

2. *apply inverse inference rule* (Inv-r) *or* (Inv-l) *to reduce the expression, and then*

3. *apply the identity inference rule* (Id-l).

**Postcondition:** *the initial term is reduced, i.e., it consists of fewer subterms.*

## 3 Method Representation

The methods we aim to learn are complex and are beyond the complexity that can typically be tackled in the field of machine learning. Therefore, we first simplify the problem and aim to learn the so-called *method outlines*. Later, we reconstruct the full information by extending outlines to methods using precondition analysis.

Let us assume the following language $L$, where $P$ is a set of primitives (which are the known identifiers of methods used in a method that is being learnt):

- for any $p \in P$, let $p \in L$,
- for any $l_1, l_2 \in L$, let $[l_1, l_2] \in L$,
- for any $l_1, l_2 \in L$, let $[l_1|l_2] \in L$,
- for any $l \in L$, let $l^* \in L$,
- for any $l \in L$ and $n \in \mathbb{N}$, let $l^n \in L$.

"[" and "]" are auxiliary symbols used to separate subexpressions, "|" denotes a *disjunction*, "," denotes a *sequence*, "$*$" denotes a *repetition* of a subexpression any number of times (including 0), and $n$ a fixed number of times. Let the set of primitives $P$ be $\{\text{A-l}, \text{Inv-l}, \text{Inv-r}, \text{Id-l}\}$. Using this language, and given the appropriate pre- and postconditions, the tactic of our simplification method described by the two examples above could be expressed as:

$$\textit{simplify} \equiv \left[\text{A-l}^*, [\text{Inv-r}|\text{Inv-l}], \text{Id-l}\right].$$

We refer to expressions in language $L$ which describe compound methods as *method outlines*. *simplify* is a typical method outline that we aim our formalism to learn automatically.

## 4 Learning Technique

Our learning technique considers some number of positive examples which are represented in terms of sequences of method identifiers, and generalises them so that the learnt pattern is in language $L$. The pattern is most specific according to the given examples, and is of smallest size with respect to some defined measure of size.

The general idea for the learning technique is (for more information see (Jamnik et al., 2001)):

- Split each example into all possible substrings.
- For each substring in each example find consecutive repetitions of inference steps, i.e. patterns.
- Find patterns that match in all examples.
- Generalise matched patterns with Kleene star or constant.
- Repeat the process on both sides of the matching pattern.
- Choose the generalisation with the smallest size.

## 5 Proof Planning Method

Method outlines do not contain all the information which is needed for the proof planner to use them. Hence, we need to restore the missing information.

A typical method that a proof planner uses does not account for repeated applications of methods (i.e., methodicals), for disjunctive applications of methods, and for termination condition for repeated applications. Hence, we build our work on the extension of the method language to provide for these constructs, as described by Jamnik et al. (2000).

Method outlines which are expressed using the language $L$ defined in §3 do not specify what the preconditions and postconditions of the methods are. They also do not specify how the number of loop applications of inference rules are instantiated when used to prove a theorem. Hence, the method outlines need to be enriched to account for these factors. We use the ideas from precondition analysis developed by Silver (1984) and later extended by Desimone (1987) in order to enrich our method representation. Precondition analysis provides explanations for proof steps. In order to be able to attach explanations to the inference rules in the style of precondition analysis, the method language needs to be extended. We extend it with the required vocabulary.

## References

C. Benzmüller, et al. ΩMEGA: Towards a mathematical assistant. In W. McCune, editor, *14th Conference on Automated Deduction*, pages 252–255, 1997.

A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer Verlag, 1988.

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, 1987. Sigma Press.

M. Jamnik, M. Kerber, and C. Benzmüller. Towards learning new methods in proof planning. In *Proceedings of the Calculemus 2000: 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, 2000.

M. Jamnik, M. Kerber, and C. Benzmüller. Learning method outlines in proof planning. Technical Report CSRP-01-08, School of Computer Science, University of Birmingham, 2001.

B. Silver. Precondition analysis: Learning control information. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning 2*. Tioga Publishing Company, 1984.