Learning Proof Methods in Proof Planning

M. Jamnik, M. Kerber and C. Benzmüller

*School of Computer Science, The University of Birmingham

THE UNIVERSITY OF BIRMINGHAM

Examples

In proof planning, often a number 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. These have to be added to the proof planning system by its developer. Our aim is to explore how a system can learn new methods automatically given a number of well chosen examples.

Inference Rules:

X o (Y o Z)	=>	(X o Y) o Z	(A-l)
e o X	=>	X	(Id-l)
X o X^i	=>	e	(Inv-r)
X^i o X	=>	e	(Inv-l)

Example 1:

a o ((a^i o c) o b)
⇓ (A-l)
(a o (a^i o c)) o b
⇓ (A-l)
((a o a^i) o c) o b
⇓ (Inv-r)
(e o c) o b
⇓ (Id-l)
c o b

Example 2:

a^i o (a o b)
⇓ (A-l)
(a^i o a) o b
⇓ (Inv-l)
e o b
⇓ (Id-l)
b

Method in pseudo-code:

Preconditions: *there are subterms in the initial term which are inverses of each other, and are separated only by brackets.*

Tactic:
1. *apply associativity (A-l) for as many times as necessary to bring terms which are inverses together*
2. *apply inverse (Inv-l) or (Inv-r) to reduce expression*
3. *apply identity (Id-l)*

Postconditions: *the initial term is reduced, i.e., it has fewer subterms.*

Method Representation

Methods that we aim to learn are complex and beyond the complexity that can be tackled in the field of machine learning. Hence, we simplify the problem and aim to learn so-called method outlines. Later we reconstruct the full information by precondition analysis.

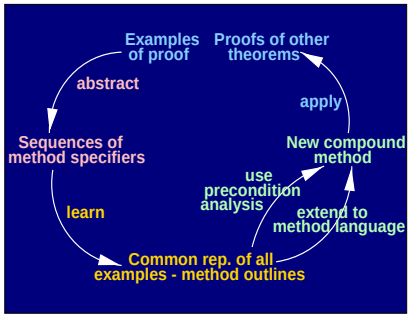
Method outlines are represented using a language L, where P is a set of primitives (which are the known identifiers of methods used in the examples; e.g., for examples above P={A-l, Inv-l,Inv-r,Id-l}):

- *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 $l^n \in L$.*

Note that: "[" and "]" are auxiliary symbols
"|" denotes disjunction
"," denotes sequence
"*" exponent denotes a repetition of a subexpression any number of times
"n" exponent denotes a repetition of a subexpression n times

Hence the method outline from examples above can be represented as::

simplify = [A-l* , [Inv-l | Inv-r], Id-l]



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.

Typical proof planning method representation: preconditions, tactic, postconditions

But: No loops (repetitions). No disjunctions.

Problem: Method outlines too abstract. No pre- and post-conditions.

Hence: Extend typical method representation.
Enrich method outlines with vocabulary and explanations (use precondition analysis).

New vocabulary for our example:

neighbour: *nb(X,Y,Z)* true is there are no terms between X and Y in Z
distance: *d(X,Y,Z)* the # of nodes -1
subterm: *subt(X,Y)*
reduced: *red(X,Y)*

Explanations:

P1: $nb(A,A^i,E)$
P2: $subt(AoA^i,E)$
P3: $d(A,A^i,E)$
P4: *subt(e,E)*
P5: *red(E,newE)*

New Learnt Proof Method **simplify:**

Pre	P1 and applicable(A-l)		
Tactic	Tact	Preconditions	Postconditions
	A-l	(P1, – or A-l) ⋮	(P1, – or A-l) ⋮
	Inv-r	(P2, – or A-l) (P1, – or A-l) (P3, – or A-l)	(P1, A-l or Inv-r or Inv-l) (P5, –)
	Inv-l	⋮	⋮
	Id-l	(P5, Inv-r or Inv-l)	⋮
Post	P5		

Learning Algorithm

Our learning technique considers a number of positive examples which are represented as sequences of method identifiers, and generalises them so that the learnt pattern is in the 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. Here is the general idea of the algorithm:

	Examples: {A-l,A-l,Inv-r,Id-l}	{A-l,Inv-l,Id,-l}
1. Split each example into all possible substrings.	1. {{[A-l],[A-l],[Inv-r],[Id-l]},{[A-l,A-l],[Inv-r,Id-l]},{[A-l],[A-l,Inv-r],[Id-l]},{[A-l,A-l,Inv-r],[Id-l]},...} {{[A-l],[Inv-l],[Id-l]},{[A-l,Inv-l],[Id-l]},{[A-l],[Inv-l,Id-l]},{[A-l,Inv-l,Id-l]}}	
2. For each substring in each example find consecutive repetitions of inference steps, i.e. patterns.	2. {{$[A\text{-}l]^2$,[Inv-r],[Id-l]},{[A-l,A-l],[Inv-r,Id-l]},{[A-l],[A-l,Inv-r],[Id-l]},{[A-l,A-l,Inv-r],[Id-l]},...} {{[A-l],[Inv-l],[Id-l]},{[A-l,Inv-l],[Id-l]},{[A-l],[Inv-l,Id-l]},{[A-l,Inv-l,Id-l]}}	
3. Find patterns that match in all examples.	3. first match: $[A\text{-}l]^2$ and [A-l]	second match: [Id-l] and [Id-l]
4. If no matches, then join patterns disjunctively.	4. not applicable	
5. Generalise matched patterns with Kleene star or constant.	5. generalise $[A\text{-}l]^2$ and [A-l] to [A-l]*	
6. Repeat the process on both sides of the matching pattern.	6. repeat on {Inv-r,Id-l} and {Inv-l,Id-l}	
7. Choose the generalisation with smallest size.	7. not applicable yet, but eventually we get	[[A-l]*,[Inv-r | Inv-l],[Id-l]]