Analogy as a Control Strategy in Theorem Proving

Erica Melis[1] and Jon Whittle*[2]

[1] Universität des Saarlandes, FB Informatik
[2] Dept. of Artificial Intelligence, University of Edinburgh

Abstract. We investigate analogy-driven proof plan construction as a control strategy in proof planning. The decisions taken in planning a *source* theorem are replayed for planning a similar *target* theorem. Our analogy procedure, ABALONE implemented on top of the proof planner *CIAM*, is capable of a controlled replay of the source planning process. In addition to the pure replay of source planning decisions, ABALONE can provide control, e.g. by overriding the default control and suggesting both induction schemes and lemmas needed in the planning process.

1 Introduction

Theorem proving by analogy is a process in which the experience of proving a *source* theorem guides the search for a proof of a similar *target* theorem. There are at least two different paradigms for analogy: transformational analogy and derivational analogy [2]. *Transformational* analogy maps a final source solution to the target and may involve additional *transformations* so the constraints of the target problem are satisfied. *Derivational* analogy instead maps the problem solving *decisions* made in the source case [3] and uses these mapped decisions as a guide to construct a target solution. That is, derivational analogy works as a control strategy in problem solving. Analogy-driven proof plan construction [8] employs derivational analogy for theorem proving, in particular in proof planning.

In this paper, we describe some ideas of the analogy system ABALONE implemented on top of the Edinburgh proof planner *CIAM* [1] which deals with

* The first author was supported by the HC&M grant CHBICT930806 and the second author by an EPSRC studentship. Computing facilities were in part provided by EPSRC grant GR/J/80702.

theorem proving by induction. CL^AM is a backward-planner. Backward proof planning starts with the conjecture as an open goal g. It *searches* for a method M applicable to g and introduces a node with M into the proof plan. (A method is a (partial) specification of a tactic where a tactic executes a number of logical inferences.) The subgoals g_i produced by the application of M become the new open subgoals and g now has status closed. The planner continues to search for a method applicable to one of the open subgoals and terminates if there are no more open goals. Methods available in CL^AM are, among others, INDUCTION, WAVE, EQUAL, and FERTILIZE. Applying INDUCTION produces subgoals that are base-cases or step-cases. Step-cases consist among others of WAVE applications that rewrite an induction conclusion until the induction hypothesis can be applied. The process of applying the induction hypothesis is realized by FERTILIZE. See Figure 2 for an example.

In this paper, we report on the realization of analogy-driven proof plan construction by ABALONE. ABALONE works as a control strategy for proof planning for inductive proofs: The proof planning process involves searching among the applicable methods. In contrast, analogy involves no search but instead methods are suggested based on past experience.

We shall mainly address the control function of analogy. Therefore, we focus on the analogical replay and neglect other features of ABALONE such as reformulations and mapping heuristics that enable more complicated analogies. (The interested reader is referred to [7] for those features.) First, the paper describes the analogical replay and the justifications used in it and then results achieved by ABALONE are given.

2 Analogy-Driven Proof Plan Construction

Analogy-driven proof plan construction works at the proof plan level. It stores planning decisions in the source along with reasons or *justifications* for these decisions having been taken. These decisions are then replayed in the target if the justifications hold in the target. Otherwise, the analogical replay suggests an alternative action.

Throughout the rest of this paper we refer to the following example:

Source theorem `lenapp`: $\forall a, b. len(a <> b) = len(b <> a)$[3]

Source induction hypothesis: $len(a <> b) = len(b <> a)$

Source induction conclusion: $len((h :: a) <> b) = len(b <> (h :: a))$

Target theorem `halfplus`: $\forall a, b. half(a + b) = half(b + a)$

Source rules:

`app2`: $(X ::_1 Y) <> Z \Rightarrow X ::_2 (Y <> Z)$

`len2`: $len(X ::_3 Y) \Rightarrow s_1(len(Y))$

`lenapp2`: $len(X <> (Y ::_4 Z)) \Rightarrow s_2(len(X <> Z))$

Target rules:

`plus2`: $s(Y) + Z \Rightarrow s(Y + Z)$

`half3`: $half(s(s(Y))) \Rightarrow s(half(Y))$.

Fig. 1. Source and target rules

Figure 1 gives the rules available for planning each of these. Some function symbols are indexed because a flexible analogy system needs mappings that can send a function symbol at different positions in the source to different target function terms. For this purpose, similarly to [6], source function symbols at different positions are differentiated by additional indices. Thereby, in our example, see Figure 1, $::_1$ from `app2` can be mapped to $\lambda w_1 \lambda w_2 . s(w_2)$ and $::_3$ from `len2` can be mapped to $\lambda w_1 . s(s(w_1))$.

[3] $<>, len, ::$ denote the append, length, and cons functions on lists.

SOURCE PLAN

$len(h ::_5 a <> b) = len(b <> h ::_5 a)$

$::_5 = ::_1$ $\quad \Big\downarrow$ wave (app2)

$\Big\downarrow$

$len(h ::_2 (a <> b)) = len(b <> h ::_5 a)$

$::_2 = ::_3$ $\quad \Big\downarrow$ wave (len2)

$s_1(len(a <> b) = len(b <> h ::_5 a)$

$::_5 = ::_4$ $\quad \Big\downarrow$ wave (lenapp2)

$s_1(len(a <> b) = s_2(len(b <> a)$

$\Big\downarrow$ fertilize

$s_1(len(b <> a) = s_2(len(b <> a)$

$s_1 = s_2$ $\quad \Big\downarrow$ elementary

TARGET PLAN

$half(s(s(a)) + b) = half(b + s(s(a)))$

$\Big\downarrow$ wave (plus2)

$half(s(s(a) + b)) = half(b + s(s(a)))$

$\Big\downarrow$ wave (plus2)

$half(s(s(a + b))) = half(b + s(s(a)))$

$\Big\downarrow$ wave (half3)

$s(half(a + b)) = half(b + s(s(a)))$

$\Big\downarrow$ wave (lemma)

$s(half(a + b)) = s(half(b + a))$

$\Big\downarrow$ fertilize

$s(half(b + a)) = s(half(b + a))$

$\Big\downarrow$ elementary

Fig. 2. Step-case replay

Figure 2 shows the step-case of the plan of **lenapp** with indexed functions.[4] We have omitted some irrelevant indices from the statement of the source theorem for the sake of clarity. In general, however, all function symbols are indexed. Dealing with indexed function symbols while running *CIAM* yields additional constraints relevant for the replay of methods, namely a sequence of so-called *C-equations* of the form $f_i = f_j$ shown in Figure 2. Note how the C-equations arise. When `app2` is applied, the LHS of `app2` matches with the LHS of the current goal - i.e. $(X ::_1 Y) <> Z$ matches with $(h ::_5 a) <> b$ which requires that $::_5 = ::_1$.

[4] The symbol $::_5$ is introduced by the INDUCTION method. ELEMENTARY applies if the current goal is easily provable using some propositional and equational reasoning.

2.1 Analogical Replay

Before the analogical replay, the indexed source and target theorem are mapped by a basic mapping m_b. m_b is then augmented by an extended mapping m_e which maps the source rules to the target rules. The mappings are designed to favour maps where the source and target are similar.

Sometimes a node by node replay is not sufficient - this could happen for instance if the source and target plans have a different number of nodes. In these cases, certain reformulations may need to be invoked during the analogical replay. Reformulations do more than just map symbols. In general, reformulations may insert, replace or delete methods or may change methods, sequents and justifications of proof plan nodes. Reformulations are triggered by irregularities of the mappings.

In the example, the basic mapping is given by $m_b(len) = half,\ m_b(<>) = +$. For the extended mapping, consider the mapping of `app2` in the example. Since $m_b(<>) = +$, the basic mapping can be used to partially instantiate `app2` to $(X ::_1 Y){+}Z \Rightarrow X ::_2 (Y{+}Z)$. The instantiation is now completed by mapping to `plus2` giving $m_e(::_{1,2}) = \lambda w_1.\lambda w_2.s(w_2)$. Similarly, `len2` maps to `half3` because of $m_b(len) = half$, giving $m_e(s_1) = \lambda w_1.s(w_1)$ and $m_e(::_3) = \lambda w_1.w_2.s(s(w_2))$. The latter violates the C-equation $::_3{=}::_2$. This violation triggers the reformulation *1to2*[5]. *1to2* changes a one-step induction in the source to a two-step in the target. In doing this, an extra constructor function is introduced into the target step-case. This means that certain source methods need to be doubled in the target.

2.2 Control by Analogical Replay in ABALONE

Guided by the source plan, the analogical replay suggests methods and thereby replaces *CLAMs* search control. The (reformulated) source plan is replayed node by node, see Figure 2. The control by analogy may deviate from the pure replay of a source proof plan. This is the case, if a justification for a source method does not hold in the target.

[5] The combination $f_i = f_j$ with $m_e(f_i)(m_e(f_i)) = m_e(f_j)$ indicates the need for a *1to2* reformulation.

The justifications for a source node are reasons for the application of the method at that node. They consist of the C-equations at that node as well as some of CI^AM method's preconditions which must be satisfied before the method can be applied. The replay of a source plan involves checking that the method justifications hold in the target. If the method justifications do not hold in the target, ABALONE may be able to establish them. This will depend on the kind of justification. We have classified CI^AM's preconditions according to whether they are legal, heuristic, or instantiational preconditions. We then decided on their role as justifications and the actions to be taken if these justifications are violated.

Legal preconditions are preconditions that must be satisfied for the method to be applied. Legal preconditions become justifications. If a legal justification does not hold in the target, it may be possible for ABALONE to establish the justification. For instance, a legal precondition of WAVE is the existence of an appropriate rule. If this justification fails in the first place because no appropriate target rule is available, an action suggests a target rule (see the example continued below).

Heuristic preconditions govern the application of methods based on a set of heurisitics. It may be possible for the analogy procedure to bypass a failed heuristic. For instance, the EQUAL method substitutes an equality $Term = Var$ in the hypothesis list into the current goal. A heuristic precondition states that Var should be a free variable. This precondition can be dropped in some cases. **zerotimes1-zerotimes3** in Table 1 is an example.[6] In the planning of **zerotimes1** x is replaced by 0 in $times(x, y)$ via the EQUAL method. However, by using m_b: $x \mapsto times(x, y)$, the image of x is no longer a free variable. By ignoring this precondition, ABALONE can still replace $m_b(x)$ by EQUAL and so it produces a plan that otherwise could not be found. Note that ignoring heuristic preconditions without the guidance of analogy is unwise.

Instantiational preconditions are preconditions that merely instantiate variables. These do not become justifications.

If a justification does not hold in the target and ABALONE cannot establish

[6] **zerotimes3** normally needs a generalisation but we can avoid this by using analogy.

or override it, a gap is left in the target plan. In this way, ABALONE may produce only partial target plans which have to be completed by base-level planning – i.e. planning without analogy.

Example continued

Consider the replay of our example (see Figure 2). ABALONE replays the INDUCTION method using the *1to2* reformulation. Then RULE(`app2`) is replayed, where `app2` was mapped to `plus2` already. *1to2* notes that RULE(`app2`) needs to be doubled in the target. This gives *two* applications of RULE(`plus2`). RULE(`len2`) is replayed easily.[7] At the RULE(`lenapp2`) node the justification fails because there is no target image for the source rule `lenapp2`. The appropriate action is to suggest a target lemma. ABALONE uses the mappings and C-equations $s_2 = s_1$ with $s_1 \mapsto s(w_1)$, and $::_4 = ::_5$ with $::_5 \mapsto s(s(w_2))$ to come up with the target rule: $half(X + s(s(Z))) \Rightarrow s(half(X + Z))$. Next, FERTILIZE and ELEMENTARY are replayed easily.

3 Results

ABALONE is implemented in Quintus Prolog as an extension to $\mathit{CL^AM}$. It has been tested on a wide range of examples. We include a selection in Table 1. In addition to these, [10] gives another 30 examples. Each entry in the table gives the source theorem in the first row and the target in the second. The third column describes whether ABALONE produces a complete plan or a partial plan with gaps that should be completed by base-level planning. For example, ABALONE produces a partial plan for **assuni-assinter** where only the base-case needs to be completed.

As a means of control, ABALONE can suggest lemmata, thereby replacing the search intensive task to speculate lemmata which is major challenge for theorem provers, see e.g. [4, 5]. Our main example **lenapp-halfplus** is an example where lemma speculations are guided by the use of analogy. [10] lists another 20 or so examples of successful lemma speculations. Analogy also suggests induction

[7] ABALONE does not double methods other than RULE(`app2`) because once `half3` has been applied, the extra constructor function has vanished.

lenapp	$len(a <> b) = len(b <> a)$	partial
halfplus	$half(a + b) = half(b + a)$	
zerotimes1	$x = 0 \rightarrow x * y = 0$	complete
zerotimes3	$x * y = 0 \rightarrow (x * y) * y = 0$	
sumrev	$sum(rev(x)) = sum(x)$	complete
evensumrev	$even(sum(rev(x))) = even(sum(x))$	
assuni	$x \cup (y \cup z) = (x \cup y) \cup z$	partial
assinter	$x \cap (y \cap z) = (x \cap y) \cap z)$	
assp2	$(y + x) + z = (x + y) + z$	complete
lenapp	$len(a <> b) = len(b <> a)$	
plussum	$sum(x) + sum(y) = sum(x <> y)$	complete
apprev	$rev(x) <> rev(y) = rev(y <> x)$	
cnc_plus	$x = y \rightarrow x + z = y + z$	complete
cnc_half	$x = y \rightarrow half(x) = half(y)$	
plussum	$sum(x) + sum(y) = sum(x <> y)$	complete
assapp	$x <> (y <> z) = (x <> y) <> z$	
sumapp	$sum(x <> y) = sum(y <> x)$	complete
prodapp	$prod(x <> y) = prod(y <> x)$	
lenapp	$len(x <> y) = len(y <> x)$	complete
doubleplus	$double(x + y) = double(y + x)$	

Table 1. Some examples run by ABALONE

schemes and thereby replaces another search intensive task in theorem proving by induction. In addition, CL^AM has a special mechanism for revising incorrect choices of induction schemes. This mechanism is also search intensive and can be applied multiple times in some proofs. We have used analogy to reduce the number of applications and hence reduce the overall search space [9].

We use analogy to override the default planner configuration. Theorem provers generally have a set library of methods to apply. Some methods, however, may be used only rarely and it could be disadvantageous to have these methods available as default. In ABALONE, unusual methods will only be applied when suggested by analogy. This can avoid divergence in the proof planner in certain cases.

As an example, consider the NORMAL[8] method. The presence of this method

[8] NORMAL removes the implication from an implicational goal and appends the ante-

is necessary to plan some theorems but is detrimental in planning other theorems. We use NORMAL only when suggested by analogy. For instance, **cnc_half** in Table 1 is such that *CLAM* can only prove it when NORMAL is present. By using analogy with **cnc_plus**, the NORMAL method can be incorporated without fear of divergence.

4 Conclusion

We have described analogy as a means of controlling the construction of proof plans for proofs by induction. We consider a node by node replay of the source plan identifying planning-related justifications which must hold for the replay to succeed. By using analogy, the search for methods is eliminated, but instead methods are suggested based on past experience.

In addition to the mere replay of source planning decisions, ABALONE can provide control, e.g. by suggesting both induction schemes and lemmas needed in the planning process. Our analogy procedure can provide control that diverges from *CLAM*s default control by alternative actions involving the attempt to establish a justification or to leave a gap in the proof plan. The analogy can provide a means of overriding default control by suggesting a configuration of methods or overriding heuristic advice.

References

1. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
2. J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. TiogaPubl., Palo Alto, 1983.
3. J.G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. Morgan Kaufmann Publ., Los Altos, 1986.

cedent to the hypotheses.

4. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.
5. D. Kapur and M Subramaniam. Lemma discovery in automating induction. In M.A. McRobbie and J.K. Slaney, editors, *13th International Conference on Automated Deduction (CADE-13)*, pages 538–552. Springer, 1996.
6. Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of ECAI-94*, Amsterdam, 1994.
7. E. Melis. Analogy in CLAM. Technical Report DAI Research Paper No 766, University of Edinburgh, AI Dept, Dept. of Artificial Intelligence, Edinburgh, 1995. available from http://jswww.cs.uni-sb.de/melis/.
8. E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 182–189, Montreal, 1995.
9. E. Melis and J. Whittle. Internal analogy in theorem proving. In M.A. McRobbie and J.K. Slaney, editors, *13th International Conference on Automated Deduction (CADE-13)*, pages 92–105. Springer, 1996. Also available from Edinburgh as DAI Research Paper No 803.
10. J. Whittle. Analogy in $CI^{A}M$. Master's thesis, Department of Artificial Intelligence, Edinburgh, 1995.