Analogy as a Control Strategy in Theorem Proving

Erica Melis
Universität des Saarlandes, FB Informatik, melis@cs.uni-sb.de
Jon Whittle
Dept of AI, Edinburgh University, jonathw@dai.ed.ac.uk

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 $CLAM$, is capable of a controlled replay of the source planning process. ABALONE provides additional control at key points of the target planning process by suggesting induction schemes, speculating lemmas, and patching failed plans.

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. Briefly, the new ideas of analogy-driven proof plan construction – compared to other analogy approaches in theorem proving – can be summarized as follows. Analogy-driven proof plan construction works at the proof *plan level*. It uses derivational analogy for theorem proving in that 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. Furthermore, it employs reformulations that produce forseeable modifications of the source plan in order to meet the target conditions.

This paper reports on the realization of analogy-driven proof plan construction by ABALONE that adds new ideas, namely a domain-dependent restriction of the mappings from the source to the target problem, definition of justifications that have to be satisfied for the analogical replay, and actions that are taken in case one of these justification is not satisfied.

ABALONE is implemented on top of the Edinburgh proof planner $CLAM$ [1] which deals with theorem proving by induction. Instead of *searching* for a method to be introduced in the proof plan, ABALONE *suggests* a method by analogy to a (reformulated) source plan. ABALONE provides additional control at key points of the target planning process by suggesting induction schemes, speculating lemmas, and patching plans. We shall mainly address the control functions of analogy. Space precludes us from describing all features of the procedure and from going into detail, e.g., about the restrictions of the mappings (Aspects of our system not covered here can be found in [7]).

The article is organized as follows. We first present some background and an example to introduce the main ideas. Then we describe the analogical replay and the control of search-intensive subtasks such as the selection of an induction scheme.

2 Some Background

$CLAM$ is a backward-planner. Backward proof plan-

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

Figure 1: Source and target rules

ning 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 an abstract 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. Inductive proof plans follow a general pattern – the method INDUCTION gives rise to base-cases consisting of symbolic evaluation and step-cases consisting, among others, of WAVE methods that rewrite an induction conclusion until the induction hypothesis can be applied by the method FERTILIZE.

ABALONE is invoked when *CLAM* cannot find a plan for a theorem in a decent time limit.

2.1 Example

Throughout the rest of this paper we refer to the following example:
Source theorem `lenapp`: $\forall a, b. len(a <> b) = len(b <> a)$[1]
Target theorem `halfplus`: $\forall a, b. half(a + b) = half(b + a)$
CLAM plans the source using primitive list induction with:
induction hypothesis: $len(a <> b) = len(b <> a)$
induction conclusion: $len((h :: a) <> b) = len(b <> (h :: a))$

Figure 1 gives the rules available for planning each of these. Function symbols in the source problem are indexed because a function symbol at different positions in the source can be send to different target function terms. Hence, we can map `app2` in Figure 1 to `plus2` by $<> \mapsto +$ and $::_1 \mapsto \lambda w_1 \lambda w_2 . s(w_2)$. We can also map `len2` to `half3` by $len \mapsto half$ and $::_3 \mapsto \lambda w_1 \lambda w_2 . s(s(w_2))$. So $::_{1,3}$ have different images.

Dealing with indexed function symbols while running *CLAM* yields additional constraints that becoe justifications, namely a sequence of so-called *C-equations* of the form $f_i = f_j$ included in Figure 2. C-equations are brought about when matching the current goal to available rewrites. 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 $::_5 = ::_1$.

Figure 2 shows the replay of the step-case of the plan of **lenapp** with indexed functions.[2] 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.

3 Analogical Mapping

In preparation for the analogical replay, the indexed source theorem is mapped to the target theorem by the *basic* mapping m_b. This mapping is then augmented by an extended mapping m_e which maps the source rules to the target rules. m_b and m_e are restricted second-order mappings that preserve a specific proof-relevant abstraction. This induction-specific abstraction guarantees the source and the target step-case sub*plans* to be identical modulo reformulations. This is a novel contribution that goes beyond known matching heuristics. In addition, a series of domain-independent evaluation heuristics was devised to rate the second-order mappings involved in m_b and m_e [10]. We prefer mappings that keep the *term structure* of the terms compared as similar as possible, e.g., by a bias against projections.

4 Analogical Replay

We replay the source proof plan node by node by applying m_b and m_e to the method at each node of the source plan and transferring these mapped methods to the target. Each source node has a justification. The node is only replayed if the mapped justification holds in the target. 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. We do not go into details of reformulations, however.

The described analogical replay of a source plan replaces *CLAM*'s search control. Justifications play a key role in that they control a justified replay and possible deviations from the planner's default control. In the following, we describe ways in which our anal-

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

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

ogy procedure overrides the default control. They are based on an analysis of the source justifications.

Justifications are reasons for the application of a method. They consist of the C-equations at a node as well as some of CL^AM method's preconditions which must be satisfied before the method can be applied. ABALONE's ability to override failure depends on the kind of justification which failed.

Legal justifications are those preconditions that must be satisfied for the method to be applied. If a legal justification does not hold in the target, it may be possible for ABALONE to establish the justification. For instance, a legal justification 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 lemma (see §3.2).

Heuristic justifications 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 states that Var should be a free variable. This condition can be dropped in some cases. **zerotimes1-zerotimes3** in Table 1 is an example.[3] 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 condition, ABALONE can still replace $m_b(x)$ by EQUAL and so it produces a plan that otherwise could not be found. Note that ignoring heuristics without the guidance of analogy is unwise.

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

By these general mechanisms CL^AM's default control can be overridden in the following way.

Lemma Speculation. Finding a lemma that will successfully complete a proof is a major and search intensive task for theorem provers [4, 5]. It is also one of the most search-intensive tasks in CL^AM. ABALONE speculates lemmas by applying m_e to lemmas or rewrites used in the source plan. Our main example **lenapp-halfplus** is an example where lemmata speculations are guided by the use of analogy – note the WAVE`(lemma)` step in Figure 2. The speculation of `lemma` is covered in §3.2. [10] lists another 20 or so examples of successful lemma speculations.

Override Default 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 method[4]. The presence of this method 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 CL^AM can only prove it when NORMAL is present. By using analogy with **cnc_plus**, the NORMAL method can be incorporated without fear of divergence.

Overriding the Default Control. During the analogical replay, it may be that mapped source justifications do not hold in the target. In some cases, however, it is possible to override this failure and take an action that will allow the target planning to continue.

Using analogy to produce a proof plan sketch. Our procedure sometimes produces a proof plan with gaps in it. We consider this an improvement on the alternative of experiencing failure in these cases. In many cases, the gaps in the plan can be filled by base-level planning. The current implementation does not provide support for automatically completing the plan in this way, but many cases can be dealt with interactively.

Suggesting Induction Schemes. Another search-intensive task in theorem proving is choosing a correct induction scheme. An incorrect choice can lead to an overly long proof or result in failure. In addition to the plan replay, ABALONE suggests induction schemes based on the mappings $m_{b,e}$. By mapping across a source induction scheme, there is no need to search through a library of schemes for the most suitable induction variable and induction term.

5 Example continued

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

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

[4] NORMAL removes the implication from an implicational goal and appends the antecedent to the hypotheses.

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]. Consider the replay of our example (see Figure 2). ABALONE replays the INDUCTION method using the *1to2* reformulation. *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. Hence, WAVE(`app2`) is replayed, where `app2` was mapped to `plus2` already and *1to2* doubles WAVE(`app2`) in the target.

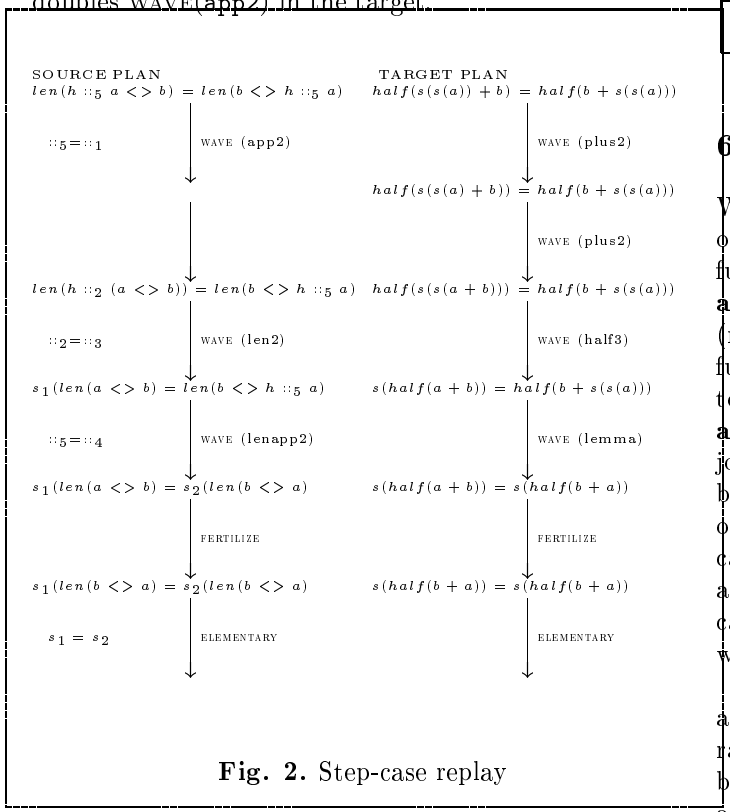

Fig. 2. Step-case replay

WAVE(`len2`) is replayed easily.[6]

At the WAVE(`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.

[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.

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

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

Table 1: Some examples run by ABALONE

6 Results

We have found examples that can be planned by analogy to a source planbut could not be proved in *CLAM* fully automatically. One such example is the proof of **assinter** by analogy with **assuni** – Table 1. *CLAM* (nor NQTHM) cannot prove **assuni** nor **assinter** fully automatically. Major user interaction is needed to complete the planning process for **assuni** and then **assinter** can be planned by analogy without any major interaction (the base case must be carried out by base-level planning[7] but this is trivial). There are other examples produced by ABALONE that *CLAM* cannot plan fully automatically. Examples (Table 1) are **evensumrev** which neither *CLAM* nor NQTHM can prove automatically and **halfplus** which *CLAM* with critics can prove but NQTHM cannot.

ABALONE is implemented in Quintus Prolog as an extension to *CLAM*. 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 (C) or a partial plan (P) 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.

7 Related Work

Our analogy has many additional features compared to Carbonell's derivational analogy [3], and it is ap-

[7]that is, planning without analogy.

plied to theorem proving. In theorem proving, the work closest to our own is that of [6] which also uses indexed functions and second-order mappings restricted by heuristics. The approaches differs, however, significantly in that PLAGIATOR [6]

- does not use derivational analogy and justifications,
- PLAGIATOR has restricted patchings compared to our reformulations. The reformulations mean that we can make analogies between theorems that differ in significant respects.
- It does not suggest induction schemes whereas we do. it is designed for equational reasoning.
- ABALONE produces proof plans, rather than proofs. This allows greater flexibility as it can override the default control if a strict analogical replay fails at some part of the proof.
- We can also yield partial proof plans that have to be completed by base-level planning, whereas [6] only produce complete proofs.
- ABALONE is embedded into a generic (proof planning) system whereas PLAGIATOR stands alone.

Other analogy systems, including [6, 9], work at the object-level. By moving to the more abstract plan level, more global control is realized.

8 Conclusion

We have described analogy as a means of controlling the construction of proof plans for proofs by induction. We consider a replay of proof planning decisions. A node by node replay of the source plan identifies 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 invoked in case of a failure of justifications. These actions involve the attempt to establish a justification, to override default control based on heuristic justifications, 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 137–162. Tioga, 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.

[4] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

[5] D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In M.A. McRobbie and J.K. Slaney, editors, *Automated Deduction CADE-13*, volume 1104 of *LNAI*, pages 538–552, Berlin, 1996. Springer.

[6] Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of 11th European Conference on Artifi-*
cial Intelligence (ECAI-94), Amsterdam, 1994. http://kirmes.inferenzsysteme.informatik.th-darmstadt.de/ kolbe/.

[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] S. Vadera. Proof by analogy in mural. *Formal Aspects of Computing*, 7:183–206, 1995.

[10] J. Whittle. Analogy in $CLAM$. Technical Report MSc.thesis, University of Edinburgh, Dept. of AI, Edinburgh, 1995.