External Analogy in Inductive Theorem Proving

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

[1] Universität des Saarlandes, FB Informatik
D-66041 Saarbrücken, Germany. melis@cs.uni-sb.de.
[2] Dept. of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK. jonathw@dai.ed.ac.uk.

Abstract. This paper investigates analogy-driven proof plan construction in inductive theorem proving. Given a proof plan of a *source* theorem, we identify constraints of second-order mappings that enable a replay of the source plan to produce a similar plan for the *target* theorem. In addition, the analogical replay is controlled by justifications that have to be satisfied in the target. Our analogy procedure, ABALONE, is implemented on top of the proof planner, CL^AM. Employing analogy has extended the problem solving horizon of CL^AM: with analogy, some theorems could be proved that CL^AM could not prove automatically.

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. Several attempts to employ analogy have been made, but these have failed to influence automated theorem proving. Early approaches, e.g. Munyer's [15], have been shown to be insufficient by Owen [16]. Approaches based on second-order matching such as [10, 6] have advanced the state of the art. They deal, however, with relatively restricted modifications of proofs only. Melis [11] describes analogy-driven proof plan construction, where proof plans are abstractions of proofs. By making analogies at the proof plan level, the potential for using analogy is extended because at the proof plan level, the analogical transfer breaks down less frequently. Furthermore, [11] introduces reformulations that map proof plans to proof plans enabling a replay between source and target plans that are significantly different in some respect.

The main issue of this paper is: how can the model of analogy-driven proof plan construction be applied to inductive theorem proving? This problem gives rise to more specific questions addressed in this paper, for instance:

- Which constraints ensure that the plan for the target theorem is similar to the source plan in inductive theorem proving?
- How can we ensure that the analogical replay yields correct planning steps in the target?

* The first author was supported by the HC&M grant CHBICT930806 whilst visiting Edinburgh and the SFB 378 and the second author by an EPSRC studentship. Computing facilities were in part provided by EPSRC grant GR/J/80702

As a basis for these investigations, we use the proof planner CL^AM [4]. Our analogy procedure, ABALONE, is implemented as an extension to CL^AM.

In contrast to [13] that addresses internal analogy, this paper investigates the more general external analogy. For internal analogy, source and target problems belong to the same theorem proving process, whereas for external analogy source and target are independent problems. In internal analogy, the source and target problems are very similar. In external analogy, complex mappings are needed to match source and target, and advanced features are needed to replay the source plan.

In the remainder of this paper we first briefly review proof planning in CL^AM (§2) as a background for the analogy. §3 introduces the analogy procedure including the mappings implemented (§3.1) and the analogical replay (§3.2). A section on our results follows.

2 Background

A proof plan for a conjecture g is an abstract representation of a proof that consists of trees of method nodes. A *method* is a (partial) specification of a tactic, represented in a meta-level language, where a tactic executes a number of logical inferences [7]. Backward proof planning as introduced by [2] 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. 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. The Edinburgh proof planner CL^AM [4] has been applied successfully to inductive theorem proving.

The main idea in CL^AM is to use a heuristic known as rippling [8, 3] to guide the search for methods. Rippling is used in the step-cases of inductive proofs. We briefly explain rippling in the following because the abstraction underlying rippling plays a major role in restricting analogical mappings and reformulations.

The major aim of step-cases in inductive proofs is to reduce the differences between the induction conclusion and the induction hypothesis so the latter can be used in the proof. To that end, CL^AM employs rippling which involves annotating the induction conclusion with *wave fronts* and *wave holes*: Wave fronts mark the differences between induction hypothesis and conclusion. Waves annotate the smallest terms containing wave fronts. Wave holes represent the parts of waves that also appear in the induction hypothesis. For example, in planning the theorem, `lenapp`:[3]

$$\forall a, b.\, len(a <> b) = len(b <> a) \tag{1}$$

the induction hypothesis is

$$len(a <> b) = len(b <> a) \tag{2}$$

and the annotated conclusion is

$$len(\boxed{h :: \underline{a}} <> b) = len(b <> \boxed{h :: \underline{a}}). \tag{3}$$

[3] $len, <>, ::$ denote the list functions length, append, and cons respectively. Typewriter font is chosen for names of conjectures.

The boxes denote the waves. Wave holes are underlined and wave fronts are the non-underlined parts within the boxes. The *skeleton* of an annotated term is constructed from wave holes and the parts of the term that do not belong to a wave. *Wave-rules* are annotated, skeleton preserving rules. E.g. a wave-rule for the function $<>$ is

$$\boxed{(X :: \underline{Y})} <> Z \Rightarrow \boxed{X :: (\underline{Y <> Z})} \tag{4}$$

where the skeleton on each side of the implication is $Y <> Z$.

Wave-rules are applied to the induction conclusion and to successive goals in the planning process. In this way, rippling moves or removes waves in the induction conclusion to a point where the induction hypothesis, represented by the skeleton of the conclusion, can be used. In our example, the waves are moved outwards. Hence, rippling provides guidance as to which wave-rules should be applied.

3 Analogy-Driven Proof Plan Construction in *CLAM*

In the following, we first explain the analogy procedure ABALONE in a nutshell. Then we go into more detail accompanied by an example.

ABALONE analogically transfers a source plan produced by *CLAM* to a plan for a target problem. The transfer is done on the basis of basic and extended mappings. These are second-order mappings which work like second-order matching but handle source constants as if they were variables. Additional induction-specific constraints restrict the mappings. The replay of the source plan is made node by node. At each node in the source we store *justifications* or reasons why a planning decision was taken. In *CLAM* these justifications tell us why a particular method was applied. During the analogy process, these justifications are also replayed. A source node is only replayed if its justifications still hold in the target. This serves two purposes. First, it guarantees the correctness of target method applications. Second, it provides a way of transferring only part of the source plan in some cases. If justifications cannot be established in the target, a gap is left in the target plan. In this way, it is only this part of the source that is not transferred, whereas the replay of the rest of the source plan can still be attempted.

Sometimes a node by node replay as described above is insufficient because the source and target plans differ in a significant respect. An example might be if the source and target plans have different induction schemes at a certain point. Rather than failing, ABALONE is equipped with a number of *reformulations* which can resolve these differences by making additional changes in the target plan. Space precludes us from describing the reformulations available in ABALONE. In this paper, we only describe the *1to2* reformulation – see §3.2. The main steps in the analogy-driven proof plan construction can be summarised as:

- Find a second-order mapping m_b from the source theorem to the target theorem.
- Extend m_b to a mapping m_e from source rules to target rules.
- Decide about the reformulations to be applied. The need for a reformulation is triggered by patterns in m_b or m_e.

- Following the source plan, analogically replay the methods. This includes
  - Apply reformulations
  - **if** method justifications hold, **then** apply method in target, **else** try to establish justification.
  - **if** justification not establishable, **then** leave gap in target plan.

Throughout the rest of this paper details of the procedure are explained and the following example illustrates the techniques.
Source theorem: `lenapp` $len(a <> b) = len(b <> a)$
Target theorem: `halfplus` $half(a + b) = half(b + a)$[4]

Source wave-rules[5]:

`app2`: $\boxed{(X :: \underline{Y})} <> Z \Rightarrow \boxed{X :: \underline{(Y <> Z)}}$

`len2`: $len(\boxed{X :: \underline{Y}}) \Rightarrow \boxed{s(\underline{len(Y)})}$

`lenapp2`: $len(X <> (\boxed{Y :: \underline{Z}})) \Rightarrow \boxed{s(\underline{len(X <> Z)})}$

Target wave-rules:

`plus2`: $\boxed{s(\underline{Y})} + Z \Rightarrow \boxed{s(\underline{Y + Z})}$

`half3`: $half(\boxed{s(s(\underline{Y}))}) \Rightarrow \boxed{s(\underline{half(Y)})}$.

The `lenapp` proof plan generated by CL^AM does not rely on the commutativity of $+$. The plan's step-case branch is displayed in Figure 1. The method WAVE applies a wave-rule, EVAL evaluates a term by applying a definition. FERTILIZE applies the induction hypothesis once rippling is completed and ELEMENTARY resolves goals that are trivially true (such as $x = x$).

Justifications Stored in the Source Plan

During the planning of the source theorem, justifications, i.e. reasons for the application of a method, are stored at each source plan node. These justifications are used later, in the analogical replay, where instead of blindly replaying the source plan, we actually replay the decisions that were taken in the source, provided the justifications for the decisions hold again in the target.

In CL^AM, each method has preconditions for its application. Legal preconditions are stored as justifications - they *must* be satisfied for the method to be applicable. The WAVE method, e.g., has preconditions that require the existence of a wave-rule that matches the current goal. Other preconditions just encode heuristics and these are not stored as justifications. In this way, our analogy procedure can override some heuristics. Overriding heuristics, without guidance from analogy, is unwise, however.

We also consider a kind of justification that is due to the use of indexed functions. ABALONE is able to send a function symbol at different positions in the source to different target images. For this reason, source function symbols at different positions are differentiated by indices. During the source planning, constraints may be placed on

[4] $half$ and $+$ denote the usual functions on natural numbers.

these indices, yielding a set of *C(onstraint)-equations* of the form $f_i = f_j$ (see Figure 1). These C-equations form an additional source justification that must be satisfied in the target for a successful replay. Since wave-rules have to be skeleton preserving, the functions that belong to the skeleton of a wave-rule have the same index on both sides of the wave-rule.

Figure 1 shows the step-case of the plan of `lenapp` with indexed functions.[6] The induction hypothesis is $len(a <> b) = len(b <> a)$. Note how the C-equations arise. When WAVE(`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 $::_5 = ::_1$.

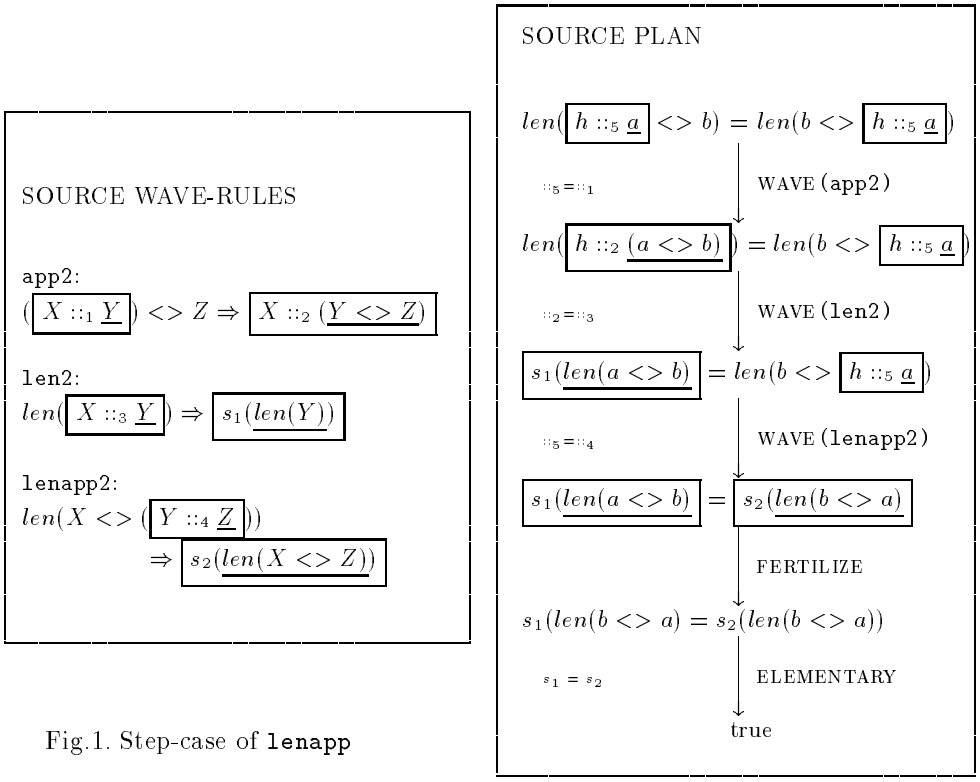

Fig.1. Step-case of `lenapp`

3.1 Mappings

We use second-order mappings, that is, mappings that send function constants to function terms, to map the source theorem to the target theorem and source rules to target rules. First, a constrained basic mapping m_b is constructed that maps the source theorem with indexed function symbols to the given target theorem. m_b is then augmented by an extended mapping m_e which maps the source rules to the target

[6] The index 5 of :: is introduced by the INDUCTION method. We have omitted indices from the statement of the source theorem for the sake of clarity because they are irrelevant in our example. In general, however, all function symbols are indexed.

rules. In addition to the most essential constraints of the basic mapping presented below, the mappings are designed to favour maps that preserve the term structure. Space prevents us from giving all the heuristics here (see [18]) but an example is biasing against projection mappings because they alter the term structure.

Basic Mapping The term tree representation of a theorem is called the *theorem tree*. The *rippling paths* in the theorem tree – indicated in bold in Figure 2 – are the paths on which the wave-fronts are moved through the theorem tree by rippling. These paths start from the induction variables.

The basic mapping m_b maps the source theorem tree ts to the target theorem tree tt and thereby it maps the rippling paths of ts to paths of tt. We try to achieve successful rippling in the target via the following induction-specific constraints for m_b that preserve the rippling paths.

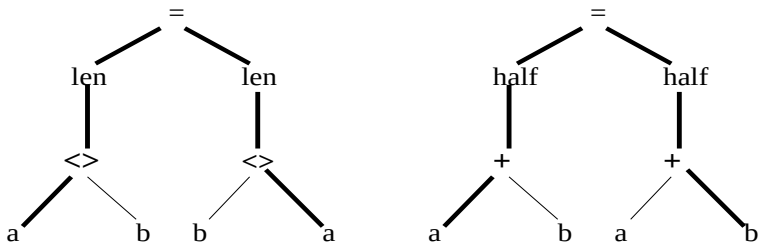

Fig. 2. Term tree representations of `lenapp` and `halfplus2` with the rippling paths in bold

Constraints for the Basic Mapping *Labelled fragments*, introduced in [9], are an abstraction of wave-rules obtained by removing the structure of the wave fronts and those parts of the skeleton not affected by wave fronts. Figure 3 displays labelled fragments of the wave-rules `len2, half3, plus2, app2`. The dots represent wave fronts. Note that in the lhs of `app2` the wave-front is situated at the first(left) argument of <> and it moves to the top of <> in the rhs of `app2`. This situation is reflected by the labelled fragment of <>.

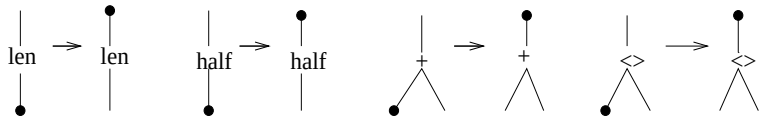

Fig. 3. Labelled fragments

The labelled fragments of function/relation nodes in a theorem tree determine the rippling paths [9]. The rippling paths in a theorem tree abstractly encode the consecutive application of the WAVE method relying on the wave-rules the labelled fragments are built from. Therefore, we require the mapping m_b to **preserve labelled fragments** or to change labelled fragments in a controlled way. The "controlled way"

gives rise to the reformulations that are beyond the focus of this paper. By this constraint, we preserve the method applications in the step-case of the source and target plans.

This constraint reduces the search space for sources and mappings. In our example, we obtain $m_b : len \mapsto half$, and $m_b : <> \mapsto +$ because the labelled fragments for len and $half$ are identical as are those for $<>$ and $+$.

Extended Mapping The extended mapping m_e provides images terms for function symbols occurring in the source plan but not occurring in the source theorem. It maps source rules to target rules. In general, function symbols in the source may have different image function terms. To some extent m_e is restricted by C-equations and by m_b. Different extended mappings are constructed for the target step-cases and for the target base-cases because different sets of C-equations belong to the step- and base-cases and different rules are used. The indexing of the rules enables different applications of the same source rule to map to different target rules.

Consider our example again. For the indexed source wave-rules refer to Figure 1 and for the target rules to §3. Since $m_b(<>) = +$, `app2` can be partially mapped. The mapping is completed by mapping the source wave-rules to available target wave rules. In this way, `app2` maps to `plus2` with $m_e(::_1) = \lambda w_1 \lambda w_2 . s(w_2)$ and $m_e(::_2) = \lambda w_1 \lambda w_2 . s(w_2)$. 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 $::_3 = ::_2$, because $::_2$, $::_3$ have different target images. This violation tells us that we need to add extra methods in the target. This is taken care of by the *1to2* reformulation described in §3.2. The source rule `lenapp2` cannot be mapped to `plus2` or `half3` by extending m_b.

3.2 Analogical Replay

As already described briefly, the main body of the analogical procedure replays the source plan node by node checking justifications at each stage. There are two occasions when ABALONE deviates from this simple node by node replay.

First, there is the case that a justification does not hold in the target. In this instance, ABALONE will try to establish the justification. Its exact action will depend on the type of justification:

- If the failed justification is associated with a WAVE method, and the situation is such that a source wave-rule has no corresponding rule in the target, then ABALONE speculates a target wave-rule. It does this by applying m_b and m_e to the source rule.
- A justification may fail because some side-condition does not hold. Whereas the side-condition may trivially hold in the source, the mapped version in the target may not hold trivially. Hence, ABALONE will set up the target side-condition as a lemma.
- Certain justifications cannot be established by ABALONE, so the source method is not replayed.

If a justification does not hold and cannot be established, then ABALONE produces a target plan node that has an empty method slot and a conclusion that contains special meta-variables, *gap variables* $?_i$. The gap variable is a place holder for the unknown subexpression of the sequent in the (current) target node that corresponds to the source subexpression that was changed by the source method which could not be transferred. Sometimes these variables can be instantiated when subsequent methods are replayed. If no subexpression other than a gap variable occurs in the current target goal, the replay stops here and proceeds to the next open target goal.

Secondly, reformulations of the source plan are needed because sometimes the mappings alone are not sufficient to produce a plan proving the target theorem. 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 in a substantiated rather than in an ad hoc way. We have come up with a number of reformulations. Each is triggered by peculiarities of the mappings or by failed justifications. We only explain the reformulation *1to2* in this paper.

In our example, the image of the source justification $::_3 = ::_2$ does not hold in the target but exposes a certain pattern triggering the reformulation *1to2*. The frequently occurring pattern consists in a combination $f_i = f_j$ with $m_e(f_i)(m_e(f_i)) = m_e(f_j)$ that indicates the need for a change of the induction scheme. *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. This is also taken care of by *1to2*, see Figure 4. In addition, an extra base-case in the target is introduced.

Let us consider the replay of the step-case of our example in Figure 4. At the INDUCTION node, ABALONE suggests the induction variable a and replays the one-step INDUCTION to a two-step INDUCTION because of *1to2*. Then WAVE(`app2`) is replayed, where `app2` was mapped to `plus2` already. *1to2* doubles the method in the target. This gives *two* applications of WAVE(`plus2`). ABALONE does not double methods other than WAVE(`app2`) because once `half3` has been applied, the extra constructor function has vanished. The next node,WAVE(`len2`), is replayed easily. At the WAVE(lenapp2) node the justification fails because there is no target image for the source wave-rule `lenapp2`. The appropriate action is to suggest a target wave-rule `lemma` by using the mappings and the C-equations to suggest a target rule. It uses the mappings plus 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 image wave-rule:
$half(X + \boxed{s(s(\underline{Z}))}) \Rightarrow \boxed{s(\underline{half(X + Z)})}$[7] As the next steps of the replay, the methods FERTILIZE and ELEMENTARY are replayed easily.

The source base-case is replayed to the first target base-case ($a = s(0)$): The first method in the base-case is EVAL(`app1`), for `app1` $:= nil <> X \Rightarrow X$. `app1` maps to the incorrect rule $s(0) + X \Rightarrow X$ because $m_e(nil) = s(0)$. ABALONE's lemma disprover spots the incorrectness, so a gap variable $?_1$ is inserted in place of $s(0) + a$ on the rhs of the target subgoal.

[7] All lemma suggestions are accompanied by a simple disprover that finds very simple false conjectures such as $x > x, F \wedge \neg F$ and which avoids unnecessary effort by rejecting some false lemmas.

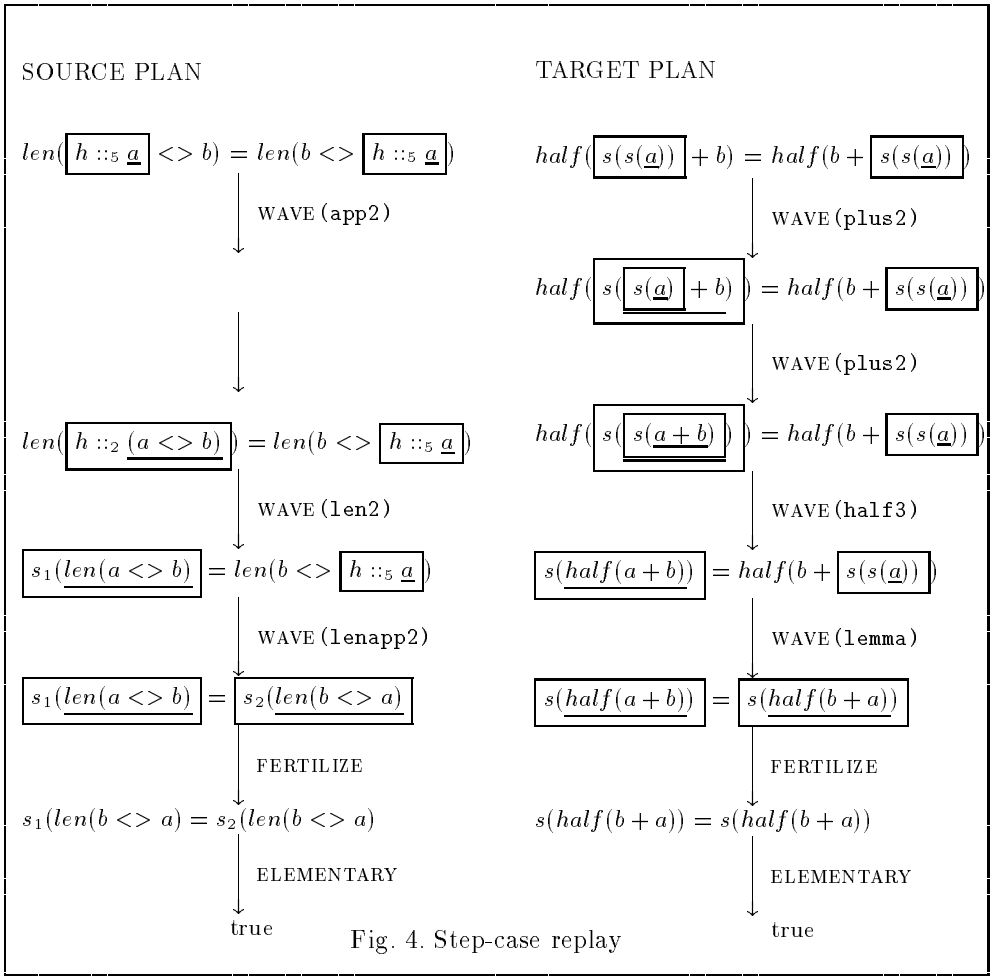

Fig. 4. Step-case replay

The replay of the base-case continues as the next method, INDUCTION, is applicable in the target. Again, the step-case of this induction is replayed with a *1to2* reformulation. WAVE(`app2`) is replayed and is doubled and WAVE(`len2`) is also replayed. The next replay of WAVE(`len2`) fails because of the presence of the gap variable where the image of `len2` would have been applied. Hence, another gap variable $?_2$ is inserted. The next method is FERTILIZE which is applied to the lhs of the target but because the induction hypothesis has $?_1$ in it, there is now a gap on both sides of the equality and so the replay stops. In this way, ABALONE has produced an incomplete proof plan (sketch) of this base-case which can be completed by base-level planning, that is, proof planning that is not guided by analogy. The other base-case is replayed by ABALONE in a similar way. Space precludes us from illustrating the base-case replay by a figure.

4 Results

ABALONE is implemented in Quintus Prolog as an extension to CL^AM. It has been tested on a wide range of examples. We include a selection in Table 1. In addition to

these, Whittle [18] gives another 30 examples. It also gives a small selection of examples on which ABALONE fails along with an analysis of why. Each entry in Table 1 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. In the `assuni-assinter` example, only the base-case cannot be obtained by analogy. This base-case is completed trivially by $CLAM$ without analogy.

Several theorems can be planned by ABALONE but could not otherwise be planned in $CLAM$ fully automatically. For example, to get $CLAM$ to prove the source `assuni`, the user has to load an uncommon method that is not available by default and has to provide lemmas for this method to use. This amounts to considerable user interaction. ABALONE replays `assuni` to produce `assinter` such that no further user interaction is required. `cnc_plus/cnc_half` is another example of this phenomenon. Although the target does not seem a difficult theorem, the presence of an uncommon method, NORMAL is required. This method is not available by default as it can create divergence in some proof planning situations. The use of analogy can provide guidance to its proper use. `zerotimes1/zerotimes3` is another example of this.

There are other examples successfully planned by ABALONE that cannot be planned by $CLAM$ fully automatically. E.g., the example `evensumrev` (Table 1) can neither be proved by $CLAM$ without user interaction nor by NQTHM [1] and `halfplus` cannot be proved by NQTHM and can be proved only by a version of $CLAM$ that is extended by a lemma speculation mechanism that is orthogonal to lemma speculation by analogy.

The strength of ABALONE comes from the fact that it replays source *decisions*. Difficult choice-points such as the choice of induction schemes and variables can be replayed. By replaying justifications, the system is flexible enough to suggest lemmas, override heuristics, and to override the default configuration of the planner.

5 Conclusion and Related Work

In this paper, we have described analogy-driven proof-plan construction in inductive theorem proving that is incorporated into a generic proof planner. As a bottom line, ABALONE can prove theorems that could not be proved by $CLAM$ or NQTHM.

The mapping and reformulation components of the system ABALONE systematically aim at providing a justified target plan that is similar to the source plan by preserving the abstractions underlying rippling or by tightly controlled changes. This paper focused on the mappings and on the role of justifications in the replay. Firstly, second-order mappings are found that satisfy certain constraints in order to have similar source and target plans. Our restriction of mappings differs considerably from other approaches, e.g., in [10]. Secondly, a node by node replay of the (reformulated) source plan produces a structured plan for the target theorem rather than just yielding the lemmas that are needed to carry out the proof as in [10]. Checking the justifications stored in the source plan ensures that the replay yields only correct planning steps in the target. Certain reactions to failed justifications try to establish justifications in the second place. Ignoring heuristic preconditions helps to speculate

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

lemmas and to override heuristic default control and default configurations. The latter changes of the control as well as the characterization of analogy as a control strategy are discussed in more detail in [14]. As outlined in §3.2, our procedure sometimes produces a proof plan with gaps in it that has to be completed by usual planning methods. 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.

These features as well as the elaborate reformulations (that are discussed very briefly only) are new compared to other approaches to theorem proving by analogy, e.g. in PLAGIATOR [10] and Mural [17]. Including the replay of induction schemes and suggestion of induction variables is also new. A further comparison with [10] shows that ABALONE works at the proof plan level and aims at transferring the proof structure rather the relevant lemmas only. Its suggestion of target lemmas is justified by mappings that preserve labelled fragments. Thereby the speculation of lemmas becomes more targeted. The main advantage of ABALONE over Mural is that it works at the plan-level rather than at the calculus-level. Thereby the analogy does not break down as quickly as the analogical transfer of calculus-level proofs. The target proofs provided by Mural are incorrect in many respects and have to be ad-hoc amended by the user, whereas our use of a justified transfer ensures that our target plan is correct.

The analogy between different source and target problems as described in this paper differs from the internal analogy presented in [13]: First of all, internal analogy works inside of planning for one theorem, where the subgoals are very similar, i.e., no reformulation and extremely simple mappings are needed. Furthermore, the current paper describes how the problem solving capabilities of a theorem prover can be extended whereas in [13] internal analogy is used for efficiency gains by replaying

very specific search-intensive subprocedures in planning. The internal analogy replays at INDUCTION nodes only and has specific justifications. External analogy is a far more difficult problem. [12] contains a comparison between experiences and goals of analogical reasoning in different theorem proving settings.

References

1. R.S. Boyer and J.S. Moore. *A Computational Logic*. Academic Press, London, 1979.
2. A. Bundy, 'The use of explicit plans to guide inductive proofs', in *Proc. 9th International Conference on Automated Deduction (CADE)*, eds., E. Lusk and R. Overbeek, volume 310 of *Lecture Notes in Computer Science*, pp. 111–120, Argonne, (1988). Springer.
3. A. Bundy, Stevens A, F. Van Harmelen, A. Ireland, and A. Smaill, 'A heuristic for guiding inductive proofs', *Artificial Intelligence*, **63**, 185–253, (1993).
4. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill, 'Experiments with proof plans for induction', *Journal of Automated Reasoning*, **7**, 303–324, (1991).
5. J.G. Carbonell, 'Derivational analogy: A theory of reconstructive problem solving and expertise acquisition', in *Machine Learning: An Artificial Intelligence Approach*, eds., R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, 371–392, Morgan Kaufmann Publ., Los Altos, (1986).
6. R. Curien, *Outils pour la Preuve par Analogie*, Ph.D. dissertation, Universite Henri Poincare - Nancy, January 1995.
7. M. Gordon, R. Milner, and C.P. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Lecture Notes in Computer Science 78, Springer, Berlin, 1979.
8. D. Hutter, 'Guiding inductive proofs', in *Proc. of 10th International Conference on Automated Deduction (CADE)*, ed., M.E. Stickel, volume Lecture Notes in Artificial Intelligence 449. Springer, (1990).
9. D. Hutter, 'Synthesis of induction orderings for existence proofs', in *Proc. of 12th International Conference on Automated Deduction (CADE)*, ed., A. Bundy, Lecture Notes in Artificial Intelligence 814, pp. 29–41. Springer, (1994).
10. Th. Kolbe and Ch. Walther, 'Reusing proofs', in *Proceedings of ECAI-94*, Amsterdam, (1994).
11. E. Melis, 'A model of analogy-driven proof-plan construction', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 182–189, Montreal, (1995).
12. E. Melis, 'When to Prove Theorems by Analogy?', in *KI-96: Advances in Artificial Intelligence. 20th Annual German Conference on Artificial Intelligence*, pp. 259-271, Lecture Notes in Artificial Intelligence 1137, Springer, (1996).
13. E. Melis and J. Whittle, 'Internal Analogy in Inductive Theorem Proving', in *Proceedings of the 13th Conference on Automated Deduction (CADE-96)*, eds., M.A. McRobbie and J.K. Slaney, Lecture Notes in Artificial Intelligence, 1104, pp. 92–105. Springer, (1996).
14. E. Melis and J. Whittle, 'Analogy as a Control Strategy in Theorem Proving', in *Proceedings of the 10th Florida International AI Conference (FLAIRS-97)*, (1997).
15. J.C. Munyer, *Analogy as a Means of Discovery in Problem Solving and Learning*, Ph.D. dissertation, University of California, Santa Cruz, 1981.
16. S. Owen, *Analogy for Automated Reasoning*, Academic Press, 1990.
17. S. Vadera, 'Proof by analogy in Mural', *Formal Aspects of Computing*, **7**, 183–206, (1995).
18. J. Whittle, 'Analogy in *CLAM*', MSc.thesis, University of Edinburgh, Dept. of AI, Edinburgh, (1995). Also available at http://www.dai.ed.ac.uk/daidb/students/jonathw/publications.html