Interleaving Processes in Case-Based Planning

Erica Melis[*] Carsten Ullrich

Universität des Saarlandes, FB Informatik
D-66041 Saarbrücken, Germany
melis@cs.uni-sb.de

Abstract. We discuss several problems of analogy-driven proof plan construction which prevent a case-based solution for more difficult target problems or make a solution very expensive. Some of these problems are due to a fixed order of the subprocesses matching, reformulation, and replay that was previously assumed and from the limited combination of planning from first principles with the analogy process. In order to overcome these problems we suggest to interleave matching and replay as well as case-based planning with planning from first principles. The mixture of case-based planning and planning from first principles is generalised to intelligently mixing different planning strategies.

1 Introduction

The common CBR cycle [1] consists of a sequence of the subprocesses matching + reformulation/ retrieval, replay, adaptation, and storage. However, empirical psychological results, e.g. in [5], question this fixed sequence of subprocesses of analogical reasoning. Computationally too we shall question the above fixed sequence of processes in analogical reasoning. In particular, we demonstrate how some problems that prevent analogy-driven proof plan construction from being more powerful and efficient can be solved by interleaving several subprocesses.

Secondly, analogical reasoning can be combined with other forms of reasoning. Psychologically this fact has been experimentally verified, e.g., in physics problem solving [14]. In several computational case-based planning systems (including analogy-driven proof plan construction) one kind of combination is realized in which planning from first principles is employed to close open subgoals remaining after the derivational replay of a source plan or to close subgoals remaining after the transfer of abstract steps, see [3]. That is, these systems use planning from first principles for the adaptation of retrieved and replayed source plans. As we shall show below, we need even more flexibility than that resulting from closing remaining subgoals by planning from first principles in order to prove theorems by analogy that are considered analogous by mathematicians.

In the remainder, we first briefly review the original analogy-driven proof plan construction. Then we discuss the problems of this and similar approaches to case-based planning and then describe solutions to these problems. Finally, we sketch a generalisation to multi-strategy planning/problem solving.

[*] This work was supported by the Deutsche Forschungsgemeinschaft, SFB 378

2 Analogy-Driven Proof Plan Construction

Analogy-driven proof plan construction was first introduced in [10] and realized in the proof planner *CIAM*[12]. Compared with previous approaches to theorem proving by analogy, its main novelties are the analogical transfer at the abstract level of proof *plans* and the need for *reformulations* of proof plans that go beyond symbol mapping. Often, these reformulations are a prerequisite for successfully second-order[1] matching a source with a target theorem that do not match in the first place. The original analogy-driven proof plan construction, as realized on top of the proof planner *CIAM* and in the ΩMEGA system [2], is outlined in Table 1.

```
input: source plan, source theorem and assumptions, target theorem and assumptions
output: (partial) target plan

Repair-match: Attempt to second-order map source and target problems triggers
reformulations of the source plan.
Reformulation: source plan ← reformulated source plan.

Replay: until source plan exhausted do
    Get next operator M from source plan.
    Check M's justifications.
    if justifications hold, then transfer M to target,
    else choose suitable action.
Plan: Plan from first principles for remaining open goals.
```

Table 1. Outline of analogy-driven proof plan construction

To start with, repair-matching tries to second-order match and when no proper match is found, an appropriate reformulation is applied to the problem and then matching is tried again. When a repair-match of the problems is obtained, the found reformulations are applied to the source *plan.*

That is, the reformulations do not only change the theorem and proof assumptions but also the proof plan. For instance, `Add-Argument` may duplicate whole subplans[2] or adding an antecedent may introduce an additional subgoal into the target plan.

Some of the frequently needed reformulations are

- symbol-to-symbol and symbol-to-term mapping,

[1] As opposed to first-order matching, second-order matching may match function and relation variables.

[2] Actually, the routine of the original `Add-Argument` reformulation that checks and changes operators and subgoals of the source plan is quite complicated, since it has to traverse and analyse the whole plan and to predict local changes.

- swapping function arguments,
- duplicating arguments of functions,
- adding an antecedent to a formula,
- freezing function arguments to constants,
- abstractions, and more complex reformulations such as
- adding/removing final or initial plan segments.

The first five reformulations have been realized in the implementations described in [12] and [11]. For instance, in ABALONE [12] the reformulation `condt` is applied when a source theorem Th_s is repair-matched with a target theorem $C \rightarrow Th_t$, where Th_s and Th_t match. `condt` introduces C as a subgoal and closes a certain plan branch, if C is disproved.

The last kind of reformulations correspond to Carbonell's T(ransformation)-operators final-segment concatenation and initial-segment concatenation [4], where a final plan segment reduces the theorem to a (set of) subgoals and an initial segment transforms assumptions to other proof assumptions. Consider, for instance, the source theorem $Th_s : \ldots \forall x.x \in setA \rightarrow x \in setB$ and the target theorem $Th_t : \ldots \rightarrow setA' \subseteq setB'$. Initially, Th_s does not match with Th_t. Only when the operator `ApplyDefinition` applies the definition of $\subseteq$ to Th_t, then the resulting subgoal matches Th_s. Of course, the additional segments can be longer than one step and more diverse than just the operator `ApplyDefinition`.

2.1 Problems with this Analogy

Experimenting with analogy-driven proof plan construction on increasingly difficult but mathematically reasonable analogy problems, led us to several problems some of which are listed below. Essentially, the discussed problems fall into four classes: (i) How to recognize the transfer of a source *sub*plan, (ii) how to realize segment concatenations efficiently, (iii) when to execute plan reformulations, and (iv) where to replay which operators?

1. The first class includes the following problems
   (a) **Subplans.** More often than not, a whole source plan cannot be transferred analogically but a *sub*plan is transferable. For example, if the source plan proves the theorem $A \wedge B$ and the target theorem B' matches with B, then only the subplan satisfying B should be transferred. Therefore, it is desirable to replay a subplan only. How do we know which subplan?
   (b) **Irrelevant matches.** Matching proof assumptions that do not belong to the replayed part of the source proof causes an unnecessary overhead.
2. **Final- and initial segment concatenation.** The search space for a sequence of reformulations that yields final- or initial-segment concatenation is potentially infinite, because potentially any sequence of plan operators has to be considered (such as `ApplyDefinition`, `Normalize`, etc). An efficient choice of a sequence of reformulations would require a severe restriction of the operators that can possibly be added and a restriction to short segments.

This is comparable with Kolbe and Walther's 'matching modulo evaluation' in [9] that allows to add one application of a definition in case the simple term mappings reformulations do not yield a match between source and target theorem. In domains with a limited choice of operators as equality proofs, this approach may be adequate as the search space is small. In more complex proofs as planned with the ΩMEGA system, this restriction is too strict.

An alternative way to realize certain segment concatenations is to prove the implication $(Th_s \rightarrow Th_t)$ – as Koehler proposed in [7] – and, if successful, add the resulting proof as an initial segment of the target plan (and similarly for the proof assumptions, add a subplan as a final segment). This alternative, however, excludes a combination with other reformulations, even with symbol or term mapping because the implication to be proved is fixed and therefore, a transfer of the plan is only possible when the source and target problem share the same predicates.

3. The third class of problems concerns the decision as to when and how reformulations should be applied.
   (a) **Reformulating before the replay?**
   When a reformulation is applied *before* the replay as realized previously, it may become pretty difficult to reformulate the source plan appropriately, because many local situations have to be distinguished. For instance, an additional conjunct C in a definition causes different changes depending on whether the definition is applied in forward planning or in backward planning. In forward planning, an `AndE` operator is introduced and the assumption C is added, whereas in backward planning the operator `AndI` is introduced and the goal C is added. Moreover the changes depend on the position at which C occurs in the target formula. For instance, it would also make a difference if C is an additional conjunct hidden in a subformula of a definition as, e.g., in $(F \equiv A \wedge C \rightarrow B)$.
   (b) **Inserting an operator or reformulating.** It is difficult to decide in advance whether the same (source) operator should be applied in the target again or whether the plan has to be reformulated first. For instance, an operator M might be applicable to a conjunctive goal as well as to a goal that is not a conjunction and in this case we do not need to apply the reformulation `AddConjunct`.
   (c) **Origins of symbols.** In case a symbol- or term mapping is needed to repair-match the source and the target theorem, it cannot generally be applied to subgoals and operators. For instance, the operator `ComplexEstimate` that is used for planning proofs of limit theorems applies, among others, the Triangle Inequality $|A + B| \leq |A| + |B|$ that contains the symbol $+$. Now if the source theorem is LIM+: $\lim_{x \rightarrow a}(f(x) + g(x)) = L_1 + L_2$ and the target theorem is $\lim_{x \rightarrow a}(f(x) * g(x)) = L_1 * L_2$, a mapping $+ \mapsto *$ would be triggered. How does this mapping affect the occurrence, change, or replacement of $+$ in a proof plan that contains `ComplexEstimate`?

As a partial solution, function- and relation symbols could be indexed. Running the source plan (again) with these indices would indicate, where which occurrence of a symbol in subgoals originates from. This indexing is realized in [8] and [12]. It may be reasonable for small proofs but produces a considerable overhead which often is unnecessary at all. Moreover, such a discrimination of symbols does not tell anything about the symbols occurring in operators.

   (d) **Plan reformulation.** A reformulation that changes the content of operators does not fit our general philosophy, *not* to act at the low level of a logical calculus but at the plan-level in case-based proof planning. Hence, in [12] reformulations of a proof plan introduce, delete, or replace operators and subgoals (as opposed to the reformulations originally experimented with in the ΩMEGA system).

4. **Corresponding goals.** Since an operator specifies a program that produces a – not necessarily fixed – sequence of inference steps, more often than not the target subgoals/assumptions to which the next operator should be applied, are not known in advance but only after the actual application of the preceding operator. Even the number of subgoals that an operator produces may depend on the planning context. Hence, one cannot decide in advance (before any operator is replayed), on which goal to replay an operator. Suppose, in the source the operator O_1 is applied to the goal g_{s1} and O_1 was replayed on a target goal g_{t1} and produced the subgoals $g_{t11} \ldots g_{t1n}$. If the next operator O_2 was applied to one of the source subgoals produced by O_1, then – because of the goal-dependency justification – it has to be replayed on one of the g_{t1i}. Hence, we have to find the corresponding subgoals. To see that this really might be a problem, consider the following example. The operator `ModusPonensBackward` is applicable to any goal. Hence, choosing the wrong goal will most likely create a false analogical transfer.

3 Solving the Problems

In this section we describe our solutions to the above mentioned problems. Table 2 gives an outline of the revised analogy-driven proof plan construction.

In the **repair-match or plan** cycle, we establish a connection between the target theorem and the source(sub)goals. First, we try to map the target theorem with the source theorem. In case no collection of *simple* reformulations[3] of the source goal yields the target theorem, we try to match with subgoals down to depth d in the source plan until a simple repair-match with a preceding subgoal can be established. If no cheap match is found, we plan one step form first principles and restart the **repair-match or plan** cycle. The planning step yields a step in an initial plan segment.

[3] Simple reformulations are: symbol mapping, term mapping, permutation of arguments, freezing arguments, duplication of arguments, adding conjuncts, adding antecedents.

input: source plan, source theorem and assumptions, target theorem and assumptions
output: (partial) target plan

```
Repair-match or plan:
    until
        a cheap second-order repair-match between target theorem and a
        source (sub)goal down to depth $d$ is found

    do plan from first principles for target goal.

$M$ := first operator, $g_s$ := source theorem

Interleaved match/replay: until source plan exhausted do

    Repair-Match:
    Find target goal $g_t$ corresponding to $g_s$ by extending the repair-match.
    if the justifications does not hold then action to establish justification.
    ReplayRef:
    if $M$ applicable then apply $M$ to $g_t$ and
    advance source plan ($M$ := next operator, $g_s$ := a goal satisfied by $M$)
    else apply reformulation triggered by repair-match.

Plan: Plan from first principles for remaining open goals.
```

Table 2. Outline of the revised analogy-driven proof plan construction

In the **match/replay** cycle, we take the next operator M from the source plan. In order to apply M in a correct way (problem 4), guided by heuristics we first find the corresponding goals by repair-matching the source goals against the target goals. When the justifications hold and M is applicable, a replay takes place. Otherwise, a reformulation triggered by the repair-match is applied (in a way corresponding to the local situation), e.g., the reformulation `AddConjunct` triggered by repair-matching source and target goals may result in additionally applying the operator `AndIntro` and in a new subgoal.

Eventually, when the source plan is exhausted, the remaining open goals are tackled by planning from first principles.

More often than not – but depending on the generality of the operators – the application conditions of an operator still hold when the repair-match consists of swapping arguments, symbol mapping, or term mapping. Then, no proper plan change is necessary at all and the reformulations are used only to specify the goal correspondences (see problem 3).

Reformulations represent heuristics on how to resolve critical differences between goals or assumptions and on how to change plans when a replay is not possible in the first place. Since reformulations change a plan by adding, removing, and replacing operators rather than by changing the (code of the) operators directly (see problem 3d), an interesting conclusion is that domain-dependent knowledge has to be used in order to determine by which operator to replace a source operator in order to be applicable to the reformulated goal. For instance, in proving theorems for real numbers that involve estimations, the correspondence between a source goal $a < b$ and a target goal $a > b$ requires to replace a $\texttt{Solve}_{<}$ operator by a $\texttt{Solve}_{>}$ operator.

3.1 What do the solutions look like?

How does this new analogy procedure solve the above mentioned problems?

Problem 1a. Going down the source plan for repair-matching means to restrict the potential replay to *sub*plans. For example, let $Th_t : A \wedge B \to C$ be the target theorem, $Th_s : F$ the source theorem and $g_{s_1} : A' \wedge B' \to C'$ a source subgoal of depth one which results from applying the operator `ApplyDefinition` on Th_s. To find the (sub)plan to be transfered, we start by repair-matching Th_t and Th_s. As no cheap repair-match is found, we continue by stepwise going down the source plan to match the subgoal g_{s_1} with Th_t. Here, a cheap repair-match is possible. As a result the subplan whose root node is g_{s_1} is replayed analogically as illustrated in Figure 1.

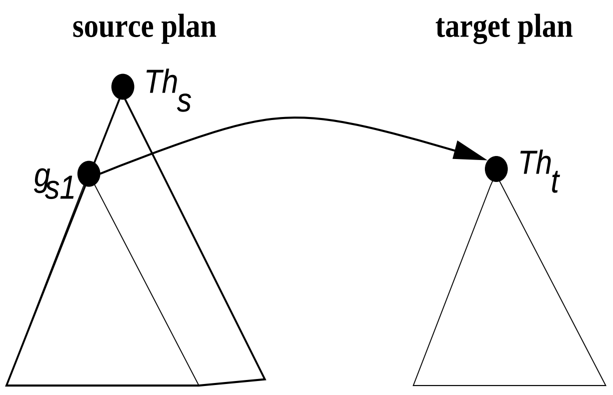

Fig. 1. Finding a subplan for the replay

Problem 1b is solved by matching assumptions when needed only. It does not make much sense to try to match every source assumption with a target assumption *before* the replay of the proof plan because not every source assumption belongs to the partial plan that is actually replayed. Therefore, this matching takes place within the **interleaved match/replay** routine.

Problem 2 is solved by the planning part in the **repair-match or plan** cycle. Consider the following example: let be $Th_s : A \wedge B \to C$ be the source theorem and $Th_t : F$ the target theorem. No cheap repair-match is found (even when looking at the subgoals), so we plan from first principles in the target. This suggests the application of the operator `ApplyDefinition` which has the subgoal $g_{t_1} : A' \wedge B' \rightarrow C'$. Now, repeating the cycle, we find a cheap repair-match between Th_s and the newly created g_{t_1}. This yields a final-segment concatenation. An initial-segment concatenation is naturally produced by planning for remaining goals including those suggested as new target lemmata by certain reformulations.

Problem 3. By delaying the application of a reformulation, we have exact information about the locally needed reformulations (e.g., is an operator applicable without any reformulation of the plan as asked in problem 3b) including the (local) effects of a reformulation that are clearly understood (thus solving problem 3a). For instance, let's assume that the last operator we replayed produced the target subgoals $g_{t_1} : A' \wedge B'$ and $g_{t_2} : C' \vee D'$. The next operator to be replayed is `ApplyDefinition`, which was applied in the source plan on $g_{s_1} : A$. Repair-matching g_{s_1} with g_{t_1} and g_{t_2} returns the cheapest match $m : g_{s_1} \mapsto g_{t_1}$ even though g_{t_1} contains an additional conjunct. Now the application of the operator `ApplyDefinition` is tried. If it is not applicable due to failing application conditions, then the failure is analyzed. The failure analysis might find that the cause is the additional conjunct B'. This would trigger a reformulation that locally introduces the additional planning step `AndIntro` with the subgoals $g_{t_3} : A'$ and $g_{t_4} : B'$. On $g_{t_3} : A'$ the replay of `ApplyDefinition` is finally possible.

Problem 4 is solved by the **interleave match/replay** cycle. Since at that point in the analogy procedure the relevant target subgoals are known from which one is to be chosen to correspond to a particular source goal.

To summarize, interleaving processes, using simple reformulations at the plan-level, and in particular, employing analogy primarily as a control strategy in planning makes the analogy easier to implement and more powerful.

4 A General Lesson: Multi-Strategy Planning

Interleaving matching and analogical replay with planning from first principles can help to reduce prohibitively large search spaces when searching for reformulations and their possible effects on a proof plan.

We generalize the interleaving of matching and replay with planning from first principles by asking: why should we solve a planning problem *only* by analogy in the first place? In fact, case-based planning can be considered a particular refinement operation in the sense of [6]. It adds steps and constraints to a partial plan and thereby reduces the set of potential solutions. Similarly, planning from first principles, the expansion of a complex operators (HTN-planning) [13], and the instantiation of meta-variables are other refinement strategies.

In the above procedure planning and match as well as match, reformulation, and replay are interleaved when needed. In the large, i.e., in a whole planning process, this can be generalized to calling the analogy by need only. This requires the (meta-)planner to have several strategies at its disposal.

Psychological evidence suggests that the unreflected analogical transfer of *all* steps of a plan, independent on how simple the subproblem is, is a rather novice-like behaviour [14].

Therefore, we suggest to represent analogical/ case-based planning as one of several problem solving strategies of a meta-planner which decides which of several strategies to use in a particular situation. Intelligently interleaving and calling several refinement operations of planning such as the introduction of operators, the expansion of operators, and the instantiation of meta-variables, can reduce the search in planning. Moreover, this makes planning more flexible and adaptable to particular situations (e.g., intelligently deciding for eager or lazy instantiation of meta-variables).

Currently, we are implementing analogy as one of several refinement strategies of the ΩMEGA planner [2]. The goal is a more intelligent, i.e. controlled, use of analogy. The control decides to call analogy, only when the analogical transfer is superior to planning from first principles.

A lesson for case-based reasoning in general is its controlled use, i.e. case-based reasoning is chosen, only when it is superior to other problem solving strategies, i.e., when it likely reduces the costs. Of course, control heuristics are necessary, e.g. heuristics that evaluate the complexity of the plan reformulations and compare this with the branching of the search space. If only one operator is applicable in a plan node (i.e., no branching occurs), then it does not make sense to choose case-based planning because it causes more overhead than planning from first principles.

Qualitative results show that multi-strategy planning is more flexible than a planner with a fix control of subprocesses. Moreover, more problems can be solved without changing the code of the planner. Moreover, new strategies can be introduced/integrated more easily into planning. For instance, when planning the theorem LIM* by analogy to the plan of LIM+, several strategies have to be employed: case-based planning, planning from first principles, and the instantiation of meta-variables.

5 Conclusion and Future Work

We presented a more flexible analogy procedure that can interleave planning from first principles with matching and that interleaves matching with replay in case-based proof planning. The greater flexibility leads to a better performance and allows to solve more problems with our analogy-driven proof plan construction.

Currently, we are implementing an even more radical interplay of case-based planning with other planning strategies that allows to invoke different strategies in one planning process. A meta-planner calls strategies according to strategic control knowledge. The meta-planner does not cause much overhead other than evaluating control heuristics.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7:39–59, 1994.
2. C. Benzmueller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge. OMEGA: Towards a mathematical assistant. In W. McCune, editor, *Proceedings 14th International Conference on Automated Deduction (CADE-14)*, pages 252–255, Townsville, 1997. Springer.
3. R. Bergmann, H. Munoz-Avila, M.M. Veloso, and E. Melis. Case-based reasoning applied to planning. In M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology from Foundations to Applications*, volume 1400 of *Lecture Notes on Artificial Intelligence (LNAI)*, pages 169–200. Springer, 1998.
4. 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.
5. K.J. Holyoak, L.R. Novick, and E.R. Melz. Component processes in analogical transfer: Mapping, pattern completion, and adaptation. In K.J. Holyoak and J.A. Barnden, editors, *Advances in Connectionist and Neural Computation Theory*, volume 2, pages 113–180. Ablex Publishing, 1994.
6. S. Kambhampati and B. Srivastava. Universal classical planner: An algorithm for unifying state-space and plan-space planning. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 61–78. IOS Press, Amsterdam, Oxford, 1996.
7. J. Koehler. Planning from second principles. *Artificial Intelligence*, 87, 1996.
8. Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of 11th European Conference on Artificial Intelligence (ECAI-94)*, Amsterdam, 1994.
9. Th. Kolbe and Ch. Walther. Second-order matching modulo evaluation – a technique for reusing proofs. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.
10. 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.
11. E. Melis. The Heine-Borel challenge problem: In honor of Woody Bledsoe. *Journal of Automated Reasoning*, 20(3):255–282, 1998.
12. E. Melis and J. Whittle. Analogy in inductive theorem proving. *Journal of Automated Reasoning*, pages –, 1999. to appear.
13. A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1977.
14. K. VanLehn and R.M. Jones. Better learners use analogical problem solving sparingly. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 338–345, Amherst, MA, 1993. Morgan Kaufmann.