When to Prove Theorems by Analogy?

Erica Melis*

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

Abstract. In recent years several computational systems and techniques for theorem proving by analogy have been developed. The obvious practical question, however, as to whether and when to use analogy has been neglected badly in these developments. This paper addresses this question, identifies situations where analogy is useful, and discusses the merits of theorem proving by analogy in these situations. The results can be generalized to other domains.

1 Introduction

Theorem proving by analogy, as sketched in Figure 1, finds a proof for a target theorem guided by a proof or proof plan of a given source theorem which is similar to the target theorem. Several attempts to implement theorem proving

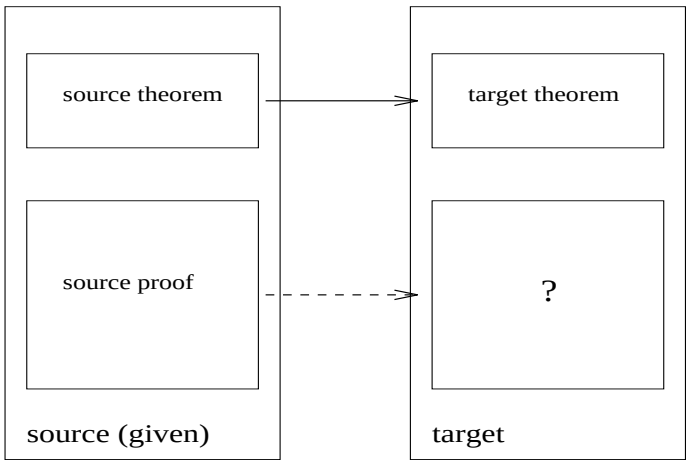

Fig. 1. Analogy in theorem proving

* The work was supported by the HC&M grant CHBICT930806 and by a grant in the SFB378

by analogy, e.g. [8, 18, 20, 9, 13], have been published. Most papers about analogy in theorem proving did refer to the well known use of analogy by mathematicians (e.g., [19]), but did not consider the actual tradeoff of automated theorem proving by analogy. On the contrary, for some approaches the storing, retrieval, and analogical replay take more time than regular theorem proving[2].

An analysis of the merits of using analogy is absolutely necessary in multi-strategy systems that are capable of both, theorem proving (without using analogy) and theorem proving by analogy. Therefore, we have to investigate how analogy pushes the problem solving horizon or improves the exploitation of the limited resources in order to evaluate the appropriateness of theorem proving by analogy. Such (limited) resources in computational theorem proving are

- Number of user interactions: Actually, user-interaction is a precious resource for interactive theorem provers that, more often than not, is used extensively.
- Run time and space: the main problem in automated theorem proving is the super-exponential search space that makes many problems intractable within limited time and space. Even some problems that appear to be easy to solve for humans who are able to structure a problem and to know good heuristics, cannot be proved automatically because of the size of the search space.
- Knowledge: Whereas too many given axioms, definitions, and lemmata blow up the search space immensely, missing axioms etc. prevent an automated theorem prover from finding a proof at all.

For problem solving in Newtonian physics, VanLehn and Jones [21] cognitively analyzed and characterized situations in which humans use analogy. They report different results for poor and good problem solvers. Similarly, we found that a distinction of different types of theorem proving systems is necessary in assessing the tradeoff in theorem proving by analogy.

Therefore, we discuss the advantages of augmenting three types of theorem provers with analogy. In the following, we investigate when to employ analogy in interactive theorem proving systems, in extensively searching automated systems, and in automated systems with little average search. We explain which advantages can be expected from the use of analogy in each type of system.

This is a paper about the experience with analogy facilities in different base systems that exhibits general principles. It is not a cognitive study, although the results resemble some findings of VanLehn and Jones as mentioned in the conclusion.

In this paper it is impossible to explain all details of these analogy facilities ANALOG, ABALONE, and internal analogy which are published in [13, 12, 15], respectively. We rather present examples of what these analogy facilities achieve.

[2] Personal communication with Christoph Walther who is an author of one of the approaches

2 Analogy in Interactive Theorem Provers: Omega

Current interactive theorem provers, e.g. Nqthm [1], require laborious user interactions. For instance, Shankar's proof of Gödel's theorem had 1741 lemmata that were formulated interactively for Nqthm. Augmenting an interactive system with an automated analogy facility naturally implies the advantage of reducing the number of user interactions and, thus, improve the efficient use of a limited resource. This applies in particular in long and complex proofs because they require many user interactions.

Take, for instance, the interactive Omega system [5], where automated theorem provers and tactics/methods can be invoked and Natural Deduction-rules can be applied. The analogy extension of its proof planner, as described in [13], works as a control strategy for the proof planner. The proof planner (interactively) produces a source plan that consists of methods, often supplied by the user.

Roughly, the analogy procedure works as follows: it higher-order matches the (parametrized) source and the target theorem and lemmata. The match triggers reformulations. Accordingly, the analogy procedure reformulates the source proof plan and suggests decisions for the choice of a (reformulated) method in the target proof plan, guided by the decisions in the source proof plan. It tests whether the reasons for this choice in the source hold in the target as well. If a target lemma is missing, lemma suggestion can yield certain reasons. Thereby it avoids the user interactions needed in order to provide the methods the target proof plan is constructed from, and to choose the right method.

In [14] we demonstrated how a user-supplied source proof plan for a Heine-Borel theorem HB1 can be transferred to a proof plan for another Heine-Borel theorem HB2, thus solving an open problem suggested by Bledsoe.

Theorem 1 Heine-Borel-1 (HB1). *If a closed interval* $[a,b]$ *of* R^1 *is covered by a family* G *of open sets (in* R^1*), then there is a finite subfamily* H *of* G *which covers* $[a,b]$.[3]

Theorem 2 Heine-Borel-2 (HB2). *If a closed rectangle* $[a,b,c,d]$ *of* R^2 *is covered by a family* G *of open sets (in* R^2*), then there is a finite subfamily* H *of* G *which covers* $[a,b,c,d]$.

In this example, the analogy procedure reduced the user interactions that provided subgoals to be proved by Natural Deduction inferences, by the automated theorem prover OTTER [11], or by a subplan from 32 to 1. Most of the HB1 proof plan was transferable by analogy as apparent from Figures 2 and 3. All reformulated source methods but `method-2'`, reformulated from `method-2`, are transferred. Only the submethod `method-21'` of the reformulated `method-2'` was transferable to the target. This left the minor target subgoal $g'5a$ to be proved by Omega.

[3] R^1 denotes the set of sets of real numbers and R^2 denotes the set of sets of ordered pairs (x,y) of real numbers x and y.

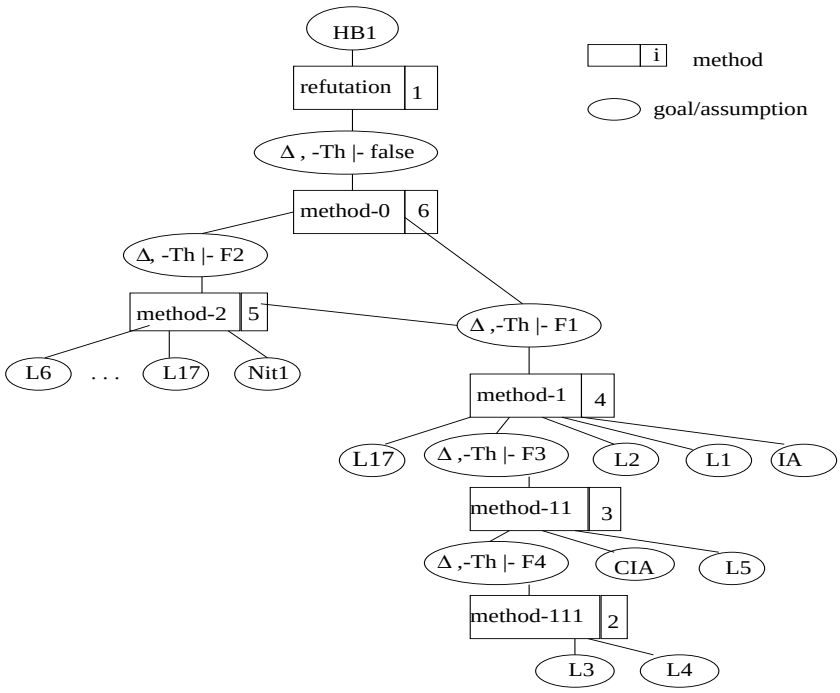

Fig. 2. The proof-plan of HB1

On the one hand, the analogical replay reduced the number of interactions. On the other hand, one additional interaction is required to provide the analogous source problem to the current analogy procedure. In case-based reasoning terms, so far the retrieval is performed by the user. This is natural though, if you remember that your maths professor told his students that they should prove some theorem B by analogy to a theorem A.

In systems that are designed as a proof assistance system, an additional argument in favor of analogy, is that an analogy facility can be a feature of human-like problem solving that contributes to the system's user acceptance.

3 Analogy in Systems with Little Average Search: *CLAM*

In this section, we record our experience with using analogy in the proof planner *CLAM*. *CLAM*, described in [3], has successfully been applied to inductive theorem proving.[4] As opposed to interactive systems, *CLAM* is an automated proof planner. It constructs proof plans that consist of methods. Some of these

[4] Induction is a generalization of Peano induction over the natural numbers that has the induction scheme $\frac{P(0), \forall k(P(k) \rightarrow P(k+1))}{\forall n(P(n))}$, where $P(0)$ is proved in the base case and

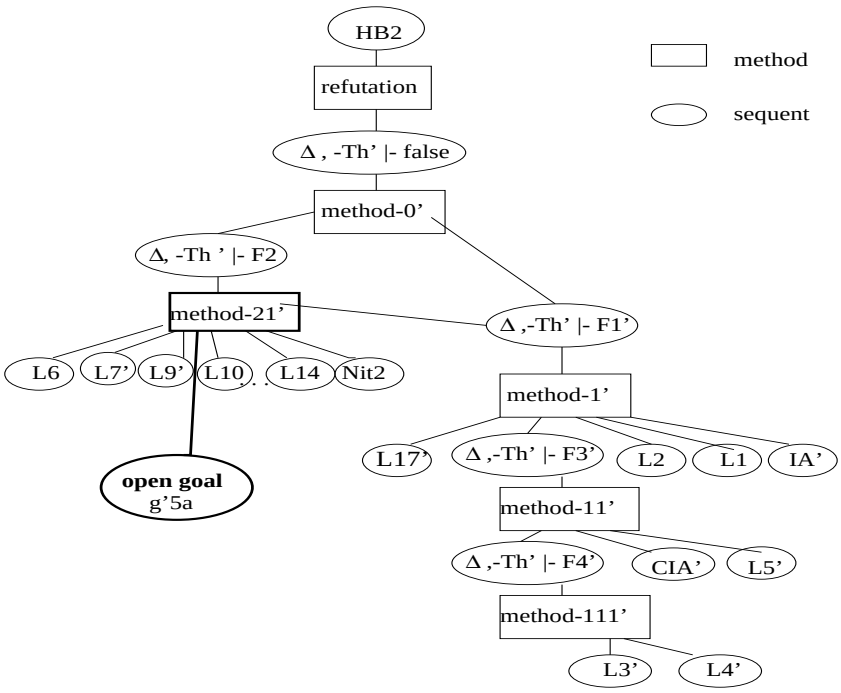

Fig. 3. The proof-plan of HB2

predefined methods are INDUCTION, WAVE, EVAL_DEF, NORMAL, and EQUAL. INDUCTION, e.g., chooses induction variables and an appropriate induction scheme and EVAL_DEF symbolically evaluates a term in the current planning goal by applying an equational definition of a function.

In CL^AM strong domain-specific search heuristics, such as *rippling* [6, 2], restrict the search for methods. Rippling systematically uses rewrites to remove differences between the induction hypothesis and the induction conclusion in a very goal-directed way so the former can be used in the proof. Because of the strong domain-specific control heuristics and because of the common plan patterns of inductive proofs[5], CL^AM is a proof planner that, opposed to most planners, typically performs *little search* for methods. For a comprehensive introduction to CL^AM see [3]. Given this behavior, usually CL^AM succeeds quickly if it masters a theorem at all.

$\forall k(P(k) \rightarrow P(k+1))$ is proved in a step case. $P(k)$ is called the induction hypothesis and $P(k+1)$ the induction conclusion. In this case, $(k+1)$ is the induction term. In the step case the induction conclusion is rewritten such that the induction hypothesis can be applied with a true result.

[5] E.g.,these proofs always consist of induction, then base case, then step case.

Therefore, the derivational analogy[6] facility, ABALONE, is invoked only if CL^AM does not succeed in a decent time limit (with the commonly loaded methods). Then the target planning process is guided by analogy to a source plan [12].

ABALONE's input is a source theorem, source rewrites, a source proof plan, a target theorem, and target rewrites. First, it incrementally produces second-mappings of the source and the target theorem and of the source and target rewrites as far as possible. These mappings of functions and relations preserve certain abstract representations of the source theorem and source rewrites that heuristically determine the proof pattern. The mappings are followed by an analogical replay of the source proof plan that includes certain reformulations of the proof plan and a check of the justifications (reasons) for the choice of the methods in the source. If a legal justification is violated in the target, no replay of the method takes place. The replay transfers a method if the failed justification is marked as "heuristic" rather than "legal". For a detailed description see [12] which is available via www. In case no given target rewrite matches some source rewrite S, the analogical mapping can suggest a target rewrite by applying the mappings to S.

The retrieval performance does not matter for the following advantages ABALONE provided because the improvements are in terms of problem solving capability rather than of speeding up. ABALONE did improve CL^AM's performance: in a situation where a lemma needed for the target proof is not given a priori, where a method not loaded by default is needed for planning a target theorem, or where other reasons e.g. the default control, prevent CL^AM from finding a plan for the target theorem within reasonable time limits. In the following, we discuss these situations for theorem proving by analogy. The given examples are simple representatives for classes of problems that CL^AM itself cannot prove but for which CL^AM with ABALONE succeeds.

- By overriding the control heuristics of the proof planner if justified by analogy, plans can be constructed which the proof planner would not find by itself. Example[7]:
  The source theorem `div3`: $\neg y = 0 \rightarrow div(plus(y, x), y) = s(div(x, y))$
  has the proof plan

```
NORMAL(...) then
  EVAL_DEF(div) then
   ELEMENTARY(...)
```

[6] Derivational analogy [4] guides the target solution by replaying decisions of the source problem solving *process*, and it uses information about reasons for the decisions (*justifications*)

[7] In the remainder of the paper s denotes the successor function, *div* division, *rev* denotes the reversion of lists, *length* the length of lists, and *app*, *cons* denote the append and cons list functions.

NORMAL places the antecedent ($\neg y = 0$) in the hypotheses list `Hyp` and then EVAL_DEF evaluates the definition of div. For the latter, a (heuristic) justification is that the antecedent `C` of the definition of div[8] is in `Hyp` (expressed by `trivial(Hyp ==> C)`). This means that EVAL_DEF(div) is applied if ($\neg y = 0$) is in `Hyp` only. ELEMENTARY recognizes the truth of the output of EVAL_DEF.
The target theorem:

$$\texttt{div3term}: \neg times(y, z) = 0 \rightarrow div(plus(y, x), y) = s(div(x, y))$$

cannot be planned by $CLAM$. The reason is that EVAL_DEF(div)'s justification `trivial(Hyp ==>` $\neg y = 0$`)` is not satisfied after the application of NORMAL. because now $\neg times(y, z) = 0$ is in `Hyp` rather than $\neg y = 0$. `div3term` can, however, be planned by analogy to `div3` that justifies to override the heuristic justification to `proveable(Hyp ==>` $\neg y = 0$`)`. Then the (correct) target proof plan is obtained

```
NORMAL(...) then
  EVAL_DEF(div) then
    SUBPROOF(not(times(y,z)=0) => not(y = 0)) then
      ELEMENTARY(...)
```

– By suggesting the use of a method that is not commonly loaded, plans can be constructed which the proof planner would not find without user intervention. For instance, the method NORMAL is not loaded by default because it often misleads the planner.
Example: The source theorem `zerotimes1`: $x = 0 \rightarrow times(x, y) = 0$ has the proof plan

```
NORMAL(...) then
  EQUAL(...) then
   EVAL_DEF(...) then
     ELEMENTARY(...).
```

NORMAL first places the antecedent ($x = 0$) in the hypotheses list and then EQUAL replaces 0 for x in the term $times(x, y)$. This is followed by EVAL_DEF that evaluates $times(0, y)$ to 0 by the definition $times(0, X) = 0$ and by ELEMENTARY that recognizes the truth of (0=0).
Among others the analogical replay automatically transfers the NORMAL method and the resulting target proof plan is

```
NORMAL(...) then
  EQUAL(...) then
   EVAL_DEF(...) then
     ELEMENTARY(...).
```

[8] which is $\neg y = 0$

– Induced by the mapping constraints, analogy can find generalizations that make a success of $CLAM$ possible at all. An example is the generalization of the target theorem

$$plus(x, plus(x, x)) = plus(plus(x, x), x)$$

to the source theorem

$$plus(x, plus(y, z)) = plus(plus(x, y), z).$$

This type of generalization is called "generalizing variables apart" and belongs to those generalizations that are difficult to choose in inductive theorem proving and, therefore, are not included into $CLAM$.

– By suggesting lemmata that are needed for the target proof target theorems can be proved which $CLAM$ itself is not able to prove. Example: Given the source theorem:

$$length(app(a, b)) = length(app(b, a))$$

the proof of which uses the source rewrite

$$length(app(a, cons(v_0, b))) = s(length(app(a, b))).$$

While analogically replaying the source proof plan for the target theorem:

$$half(plus(a, b)) = half(plus(b, a))$$

the analogy facility suggests the target lemma:

$$half(plus(a, s(s(b)))) = s(half(plus(a, b)))$$

by applying the mappings established between the source and the target and other constraints.

4 Analogy in Systems with Extensively Searching Sub-routines: *CLAM*3

When working on analogy in $CLAM3$, we identified another situation where analogy did improve the system's performance. $CLAM3$ is a version of $CLAM$ that is extended by *critics* which are very search-intensive procedures that help to continue the proof planning when $CLAM$ itself gets stuck.

Critics [7] are an extension of proof planning that patch failed proof attempts. For instance, it can happen that INDUCTION selects an inappropriate induction scheme. Then planning using an incorrect scheme becomes blocked. The *induction revision critic* patches an incorrect choice of the induction scheme by introducing extra function variables into the induction term so that rippling can continue and gradually instantiates the function variables by higher-order matching with given rewrites. The instantiation requires to go back to the INDUCTION node and to suggest a revised induction term.

For instance, consider the conjecture

$$even(length(app(t,l))) = even(length(app(l,t))) \quad (1)$$

INDUCTION originally suggests an induction on l because of the existence of a certain rewrite. This gives an induction conclusion of

$$even(length(app(t,cons(h,l))) = even(length(app(cons(h,l),t)))$$

Applications of rewrites eventually yield the subgoal

$$even(s(length(app(t,l)))) = even(s(length(app(l,t))))$$

No further rewrite is applicable, so the planning process is blocked. The way the induction revision critic deals with this is to introduce function variables into the induction term the instantiation of which yields a revised induction term $cons(h_1,(cons(h_2,l)))$ with which the planning succeeds.

It can happen that within the same proof, the induction revision critic is called a number of times. The internal analogy avoids such waste of time. *Internal analogy* is a process that transfers experience from a completed (source) subgoal in the *same* problem to solve the current (target) subgoal. That is, internal analogy works on similar subproblems of a single problem.

For instance, the planning of the conjecture

$$even(length(app(x,y))) = even(length(app(y,x)))$$

includes four induction revisions from a one-step list induction to a two-step list induction. In this situation the internal analogy overrides the default control of CL^AM3 at INDUCTION nodes in proof planning by overriding INDUCTION's default choice with an instantiation of the result of a prior application of the induction revision critic. Note that the internal analogy has been interleaved with regular proof planning.

On the one hand, the additional effort to store the relevant information is small for the particular internal analogy. The information that an induction revision took place, which subterm was affected by the critic application, and the previously revised induction scheme is stored in the respective INDUCTION nodes during the planning process. In addition, no or very simple mapping is necessary. The previous INDUCTION nodes have to be checked for critic applications in our procedure. On the other hand, by reducing the number of critic calls, search can be reduced in the following ways:

- In CL^AM3, one of the most intensive sub-procedures is the search for the induction scheme. The internal analogy facility suggests an induction scheme so the search for these is eliminated.
- The effort needed to actually apply the induction revision critic again, in particular the expensive higher-order matching, is eliminated.
- The critic is not applied until CL^AM3 has already chosen an incorrect induction rule and continued to the point where further planning is blocked. Internal analogy removes this backtracking altogether.

Hence, the savings by internal analogy outweigh the additional effort needed. The retrieval is included into the additional effort.

What about the additional effort in cases where no use can be made of analogy? In one kind of cases, different types of induction revision can occur in the same proof. If there is no map available from the stored justification, an incorrect induction revision is *not* imposed, and a second critic has to be applied as usual. Obviously, time is taken to look for a mapping from the stored justification to the current goal, which is redundant because the second revision has to be carried out anyway, but the computational cost is low.

The same happens in those cases where no critic is needed for a subsequent induction. Then no map from the stored justification will succeed, so the induction is not altered.

In a third kind of cases it happens that once a revision has taken place, a mapping from the justification to a future goal is successful, even though INDUCTION would correctly choose the induction scheme. So even though no critic application is avoided, the analogy can still be used to suggest an induction scheme. Even though analogy is not needed in this case, it is difficult to prevent analogy from being applied. This is not disadvantageous, however, since there is still a saving in time here, because the need to search for the induction variable and term is eliminated.

Some test results are given in Table 1. As expected, the costs associated

Conjecture	T1	T2
even(length(app(x,y)))=even(length(app(y,x)))	708	567
half(length(app(x,y)))=half(length(app(y,x)))	329	295
even((plus(x,y))=even(plus(y,x))	48	40
even(length(app(x,y)))=even(plus(length(y),length(x)))	107	96
half(plus(x,y))=half(plus(y,x))	48	45
even(length(x))=even(length(rev(x)))	141	122
even(half(plus(x,y)))=even(half(plus(y,x)))	2961	2650
half(length(x))=half(length(rev(x)))	55	53
even(plus(z,length(app(x,y))))=even(plus(z,length(app(y,x))))	3850	3502

T1 *is the time CLAM3 needs to plan the given theorem without using internal analogy.* T2 *is the time it takes with internal analogy.*

Table 1. Some examples run by our system

with the internal analogy - storing the justification, comparing the justification with the subgoals at subsequent INDUCTION nodes, and suggesting the induction schemes - turned out to be less than those associated with the application of the induction revision critics. As a result, the time taken to prove the theorems given was reduced. As well as the gain in runtime, analogy makes the proof planning process clearer because the redundant part of the original proof caused by incorrect selection of induction schemes can be eliminated.

5 Conclusion

What can we learn from this experience? To make a long story *very* short,

- the use of analogy in interactive systems has several advantages;
- in systems that do little search, use analogy as a last resort to solve problems that cannot be solved otherwise;
- use analogy if it can replace a search-intensive subroutine at low cost;
- only if none of the preceeding criteria applies, then an empirical or theoretical analysis might reveal conditions for an efficient use of analogy for classes of problems.

In a little more detail and including related work, the story is summarized as follows:

1. For interactive systems
   (a) some user interaction can be replaced by an automated effort for the analogical transfer of a source proof or proof plan. With respect to the resource "human interaction", the bias is always in favor of analogy, in particular for long or complex solutions. This is independent of the complexity of the analogy procedure. Even if the retrieval is done interactively, user interaction is saved.
   Similarly, Reif and Stenzel [20] report substantial savings in software verification when a reuse facility is integrated into their system. This is because user interaction accounts for the lions share of formal software verification.
   (b) In assistance systems, analogy can be a feature of human-like problem solving that contributes to the system's user acceptance. Again, this argument in favor of analogy is orthogonal to any complexity argument.
2. For automated systems with extensive search, time is a resource that can be saved.
   (a) Analogy can save search by analogically replaying a source proof or proof plan. Here, the time taken for retrieval has to be taken as part of the (time) effort needed for analogy and the savings have to outweight the additional effort. Veloso [22] compares run times of regular problem solving vs. problem solving by analogy for domains or problems for which the problem solving involves a lot of search. Presumably, these results naturally transfer to theorem proving by analogy although no empirical tests have been conducted so far. For practical purposes, a worst case complexity analysis as in [17] will not do the job.
   This situation compares to the empirical results of VanLehn and Jones about poor physics problem solvers who display many episodes of analogical problem solving and use analogy instead of regular problem solving even when this is not most effective.
   (b) For specific search-intensive procedures internal analogy can save search at low costs, as explained in section 4. In case the internal analogy does not hold, the additional costs are low.

3. For automated systems with little average search, an analogy procedure is invoked when the base system is stuck (cannot prove the theorem from the given assumptions in a decent time limit). This means that analogy pushes the problem solving horizon of the base system. This fact amounts to a bias in favor of analogy independently of any complexity analysis including the retrieval. A system augmented by derivational analogy can solve more problems than the base system itself
   (a) by overriding the default control of the base system for the particular target or by replaying an uncommon system configuration. Again, this "learning" of control knowledge compares to the empirical results of VanLehn and Jones for good problem solvers (who have little average search) who learn search control by analogy.
   (b) by providing originally missing knowledge (lemmata) needed to solve the target problem. This also compares to empirical results of VanLehn and Jones for good physics learners who use analogy to fill knowledge gaps. Apart from analogy, other learning techniques such as EBL [16] and chunking [10] can provide control knowledge. We do not discuss them here.

6 Acknowledgment

I would like to thank Alan Bundy who influenced my struggle with *CLAM* considerably by asking 'what does analogy buy?'. Thanks to Jörg Siekmann and Wolf Schaarschmidt for reading drafts of this paper.

References

1. R.S. Boyer and J.S. Moore. *A Computational Logic Handbook*. Academic Press, San Diego, 1988.
2. A. Bundy, Stevens A, F. Van Harmelen, A. Ireland, and A. Smaill. A heuristic for guiding inductive proofs. *Artificial Intelligence*, 63:185–253, 1993.
3. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.
4. 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.
5. X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Omega-MKRP: A Proof Development Environment. In *Proc. 12th International Conference on Automated Deduction (CADE)*, Nancy, 1994.
6. D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *Proc. of 10th International Conference on Automated Deduction (CADE)*, volume Lecture Notes in Artificial Intelligence 449. Springer, 1990.
7. A. Ireland and A. Bundy. Productive use of failure in inductive proof. Technical report, Department of AI Edinburgh, 1994. Available from Edinburgh as DAI Research Paper 716.

8. R.E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2:147–178, 1971.
9. Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of ECAI-94*, Amsterdam, 1994.
10. J Laird, A. Newell, and P. Rosenbloom. SOAR:an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
11. W.W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.
12. 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/.
13. 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.
14. E. Melis. Theorem proving by analogy – a compelling example. In C.Pinto-Ferreira and N.J. Mamede, editors, *Progress in Artificial Intelligence, 7th Portuguese Conference on Artificial Intelligence, EPIA'95*, Lecture Notes in Artificial Intelligence, 990, pages 261–272, Madeira, 1995. Springer.
15. E. Melis and J. Whittle. Internal analogy in inductive theorem proving. In *Proceedings of the 13th Conference on Automated Deduction (CADE-96)*, LNAI, Berlin, New York, 1996. Springer. also published as DAI Research Paper 803.
16. T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.
17. B. Nebel and J. Koehler. Plan reuse versus plan generation. a theoretical and empirical analysis. *Artificial Intelligence*, 1995. Special Issue on Planning and Scheduling.
18. S. Owen. *Analogy for Automated Reasoning*. Academic Press, 1990.
19. G. Polya. *How to Solve it*. 2nd ed. Doubleday, New York, 1957.
20. W. Reif and K. Stenzel. Reuse of proofs in software verification. In R.K. Shyamasundar, editor, *Proc. 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *LNCS*. Springer, 1993.
21. 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.
22. M.M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer, Berlin, New York, 1994.