Internal Analogy in 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@aisb.ed.ac.uk.

Abstract. Internal analogy tries to reuse solutions of subproblems within the same problem solving process. In mathematical theorem proving several patterns are known where internal analogy suggests itself. Hence, we propose the use of internal analogy in automated theorem proving. This paper investigates the possibility of incorporating internal analogy into the inductive proof planner *CIAM*. It introduces internal analogy as a control strategy of *CIAM* that can reduce search. We concentrate on using internal analogy to avoid the repeated application of the induction revision critic. The implementation has been tested and it was found that internal analogy can reduce the time taken to construct proof plans for some theorems.

1 Introduction

Analogy in problem solving transfers the solution or problem solving experience of a source problem to guide the problem solving for a similar target problem. *Internal analogy* is a process that transfers experience from a completed subgoal (source) in the *same* problem to solve the current subgoal (target). That is, internal analogy works on similar subproblems of a single problem.

Limited research has been conducted on internal analogy. In [Kennedy 95, Hickman & Larkin 90], internal analogy has been studied for a series of physics problems. This research has shown that the problem solution or the problem solving process has to be well structured in order to enable the application of internal analogy.

We consider the application of analogy to theorem proving. In particular, internal analogy in theorem proving reuses the solutions of previously solved subtasks to help in finding the solution to other subtasks within the same proof. Thereby, internal analogy can offer an additional means of search control. As far as we know, internal analogy has not yet been considered in theorem proving, apart from a very simple internal analogy for the verification of modified programs [Reif & Stenzel 93]. In mathematics, however, several patterns are known where internal analogy suggests itself. For example, in proofs of theorems of the

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

form $(A \leftrightarrow B)$ the subproofs of each single conjunct of $(A \rightarrow B \wedge B \rightarrow A)$ are often very similar.

This paper addresses the possibility of incorporating internal analogy for a particular subtask into the inductive proof planner *CLAM* producing proof plans [Bundy *et al* 91]. A *proof plan* is an abstract representation of a proof that consists of a tree of *methods* that are (partial) specifications of object-level tactics [Gordon *et al* 79]. Proof plans for inductive proofs tend to be well-structured. In addition, inductive proofs often involve a number of syntactically similar subgoals. Hence, a proof planning system is a well-suited environment for the integration of internal analogy.

*CLAM*3 is an extension of the basic *CLAM* system that uses *critics* (see [Ireland & Bundy 94]) to patch certain types of failed proof. In particular, the *induction revision* critic patches an incorrect choice of the induction scheme. Many proof plans involve multiple applications of this critic. We have extended *CLAM*3 by incorporating the use of internal analogy to avoid the repeated use of the induction revision critic.

In the remainder of this paper we first give some necessary background about *CLAM*3's search control and critics[3]. Then we present the use of internal analogy to avoid multiple applications of the induction revision critic along with a discussion of the additional effort needed and the search saved. Finally, the implementation and results show the overall savings gained by this type of internal analogy.

2 Background

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, *CLAM* employs domain-specific search heuristics restricting the search for methods, most notably *rippling* [Hutter 90, Bundy *et al* 93]. Rippling involves annotating the induction conclusion with *wave fronts* and *wave holes*. Wave fronts mark the differences between induction hypothesis and conclusion. Wave holes represent the parts of waves that also appear in the induction hypothesis. A particular wave front together with its associated wave holes is called a *wave*. For example, in planning the goal

$$\forall x, y, z.\; x + (y + z) = (x + y) + z \tag{1}$$

the induction hypothesis is

$$x + (y + z) = (x + y) + z \tag{2}$$

and the conclusion is

$$\boxed{s(\underline{x})} + (y + z) = (\boxed{s(\underline{x})} + y) + z \tag{3}$$

[3] If a remark applies to both, *CLAM* and *CLAM*3, we write *CLAM*

where s is the successor function for natural numbers. 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[4], skeleton preserving rewrites. E.g., a wave-rule for the function $+$ is

$$\boxed{s(\underline{U})} + V \Rightarrow \boxed{s(\underline{U + V})} \tag{4}$$

where the skeleton on each side of the implication is $U + V$.

Rippling works by applying *wave rules* to the induction conclusion and to subgoals in the plan. By applying wave rules to successive goals in the planning process, wave fronts can be moved or removed. In this way, rippling works towards its goal of moving those parts of the induction conclusion which are marked by the wave fronts, so that the induction hypothesis, represented by the skeleton, can be used. The repeated application of (4) to goal (3), for instance, moves the wave fronts outwards until the following expression is obtained

$$\boxed{s(\underline{x + (y + z)})} = \boxed{s(\underline{(x + y) + z})} \tag{5}$$

Now the induction hypothesis can be used to rewrite this expression.

2.1 Critics

Critics [Ireland & Bundy 94] are an extension of proof planning, whereby the failure of methods is used to patch failed proof attempts. We shall give a description of the *induction revision* critic, as its use is integral to the work presented in this paper. *CλAM*'s induction method chooses an induction scheme for the problem at hand according to which scheme will enable rippling to take place. It can happen that the induction method selects an inappropriate induction scheme. As a result, the planning process becomes blocked - i.e. it reaches a point where no further methods can be applied so the planning fails. It is at this point of blockage that the induction revision critic is invoked to revise the choice of induction scheme. Consider the conjecture[5]

$$\forall t, l : list(obj).\ even(length(t <> l)) = even(length(l <> t)) \tag{6}$$

where $even(X)$ is true when X is even, for a natural number X. $length(X)$ gives the length of list X. $X <> Y$ is the result of appending the list X onto the list Y. Suppose the following wave rules are available:

$$\boxed{U :: \underline{V}} <> W \Rightarrow \boxed{U :: \underline{(V <> W)}} \tag{7}$$

$$even(\boxed{s(\underline{\underline{\boxed{s(\underline{U})}}})}) \Rightarrow even(U) \tag{8}$$

[4] The annotations being the wave fronts and wave holes

[5] This example is taken from [Ireland & Bundy 94].

$$length(\boxed{U :: \underline{V}}) \Rightarrow \boxed{s(\underline{length(V)})} \tag{9}$$

$$length(U <> \boxed{V :: \underline{W}}) \Rightarrow \boxed{s(\underline{length(U <> W)})} \tag{10}$$

where :: is the infix *cons* operator. Wave rule (7) contains the subterm $\boxed{U :: \underline{V}}$ on its LHS and this motivates the choice of a one-step induction with the induction term $\boxed{h :: \underline{l}}$ and the induction variable l by the induction method. This gives an induction conclusion of

$$even(length(t <> \boxed{h :: \underline{l}})) = even(length(\boxed{h :: \underline{l}} <> t)) \tag{11}$$

Applications of wave-rules (7), (9), (10) eventually yield the subgoal

$$even(\boxed{s(\underline{length(t <> l)})}) = even(\boxed{s(\underline{length(l <> t)})}) \tag{12}$$

No further method is applicable, so the planning process is blocked. The call of the induction revision critic is motivated by a blockage characterized by the absence of a matching wave rule but the presence of a *partial* wave rule match. We say that there is a partial match between two terms t_1 and t_2 if the terms cannot be matched but if a match can be achieved by introducing meta-variables into t_1 to produce t_1' which can then be matched with t_2 by instantiating the meta-variables suitably. In the example, the critic looks for a partial match of a wave rule with one of the blocked terms in (12). There is a partial match between either side of the equality in (12) and wave rule (8) because we can introduce meta-variables F_1, F_2 into (12) giving:

$$even(\boxed{s(\boxed{F_1(\underline{length(Y <> \boxed{F_2(\underline{Z})})})})})^{6} \tag{13}$$

These meta-variables can be instantiated (F_1 to s(..), F_2 to the identity function) to provide a match with the LHS of (12). The critic then goes back to the induction method and suggests a new induction term of $\boxed{h_1 :: h_2 :: \underline{l}}$. It discovers this new term by rippling-in[7] on the instantiated (13), using wave rules (7) and (9) from right to left. This eventually gives

$$even(length(\boxed{h_1 :: \underline{\boxed{h_2 :: \underline{l}}}} <> t)) \tag{14}$$

which suggests the revised induction term $\boxed{h_1 :: h_2 :: \underline{l}}$ with which planning succeeds.

[6] The dotted lines represent potential wave fronts - they may or may not become genuine wave fronts.

[7] Rippling-in moves wave fronts inwards rather than outwards.

3 Internal Analogy in CL^AM

The process of reasoning by analogy can be described as follows. A case base is kept of previously solved problems with accompanying solutions. When a new problem is encountered, a similar problem is retrieved from this case base and its solution is used as a guide to the solution of the new problem. As the case base can become very large its entries have to be indexed in some way to make the retrieval efficient. In addition, analogy requires to map, and sometimes to reformulate, the source problem to the target problem and to extend the mapping and reformulation to the solutions.

Before we describe the actual internal analogy control strategy, we want to justify the use of internal analogy in more general terms. We do this by identifying those situations in which a problem solver benefits from analogy and show which situation applies to our internal analogy. Such an analysis can provide the reader with useful arguments on when it is possible to use analogy efficiently in their own problem solver (theorem prover).

Employing analogy is beneficiary in one of the following three kinds of situations:

1. Compared to base-level problem solving (planning, theorem proving) analogy can save search considerably exceeding the additional effort that is necessary to run the analogy.
2. User interaction is saved by applying analogy.
3. The integration of analogy into the problem solver makes it possible to solve more problems.

Internal analogy, as presented below, gives rise to the first situation in which analogy should be employed There, the case base of stored subtasks will remain very small because only information about subtasks within the current problem needs to be stored. No indexing is necessary. The retrieval procedure in internal analogy is also very simple because the case base is small and so the number of candidate source problems is kept low. Finally, the mapping effort involved is minimal because the similar subtasks of one problem share many features.

As a result, the effort needed to execute internal analogy is relatively low so a considerable reduction of the overall effort can be achieved. if analogy replaces search-intensive procedures. In CL^AM3, for instance, the application of the induction revision critic is a very search intensive subtask. Hence, avoiding repeated applications of this critic is a candidate for employing internal analogy.

3.1 Avoiding Repeated Critic Calls

We now describe our technique for avoiding the repeated application of the induction revision critic in CL^AM3. It is often the case that within the same proof, the same induction revision is made a number of times. For example, the conjecture

$$even(length(x <> y)) = even(length(y <> x)) \tag{15}$$

makes four induction revisions from a one-step list induction to a two-step list induction. As [Ireland & Bundy 94] says, these types of problems "fall into a class of inductive theorems that are not uncommon but are acutely difficult to prove". Hence, it is worth considering techniques that reduce the time taken to prove such theorems.

In a proof planning process, we allow the first occurrence of a critic to proceed as normal. Then, at any following induction nodes, if possible, we suggest an induction scheme analogous to the scheme suggested by the initial critic application, thus bypassing further critic applications. This procedure reduces search in the following ways:

- The analogy suggests an induction term and induction variable so the search for these is eliminated. Although, usually little search is involved in CL^AM, one of the most intensive aspects is the search for induction term and variable. This is because CL^AM has to search through the potential terms to see which will allow rippling to take place and also through possible induction variables, choosing the one with most occurrences involved in rippling.
- The effort needed to actually apply the induction revision critic again is eliminated. There are two main facets to this - higher order matching of a schematic term and the wave rules available, and rippling-in on the instantiated schematic term. By eliminating this step, we eliminate the effort of rippling-in (which involves search as to which wave rules to apply) and the higher order matching which is expensive.
- The critic is not applied until CL^AM3 has already chosen an incorrect induction rule and rippled to the point where rippling is blocked. Hence, the search needed to produce this redundant rippling is eliminated. In its original form, once rippling becomes blocked, CL^AM3 has to backtrack to the previous induction node, where the induction scheme is changed. Internal analogy removes this backtracking altogether.

We use our particular internal analogy as a control strategy that can override CL^AM3's default choice of the standard induction method. There follows a brief explanation of Figure 1 while more details can be found later in this section. The general idea is that CL^AM3 proceeds as normal, applying methods, until an induction node is reached. If no induction revision critics have been applied so far, CL^AM3 chooses the induction scheme in the usual way and continues planning. It may happen that this scheme has to be revised by a critic - in which case, the *justification* (reason) for the revision and the revised scheme are stored. The revision is carried out and planning continues. If at any stage, CL^AM3 plans an induction node and there is a stored justification of an induction revision, the system will attempt to discover if the justification holds in the current (target) goal. If so, the induction scheme will be suggested, using analogy rather than running the induction method.

The procedure that includes CL^AM3's planning and internal analogy is shown in Figure 1. $refine\_goal$ attempts to plan a proof for the given goal. Given a previously stored induction scheme, $map\_revision$ suggests a new induction scheme for the current goal by analogy. As standard in CL^AM3, $choose\_method(G)$ finds a method $Mthd$ that is applicable to the current goal, G. $apply\_method(Mthd, G)$ applies $Mthd$ to the current goal G yielding new subgoals $SubGoals$. $critic\_applicable(G)$ succeeds if the induction revision critic can be applied. Note that the application of critics arises only if no further methods can be used (i.e. if planning is blocked).[8]

$refine\_goal(G)$

input: Goal G Output: Plan for G

> **If** $choose\_method(G) = Mthd$ **then**
>> **if** $Mthd = induction(\_)$ **and** $stored\_revision(Scheme, Term)$ **and**
>> $map\_revision(Scheme, Term, G) = NewScheme$
>> **then** $apply\_method(induction(NewScheme), G) = SubGoals$ **and**
>> $refine\_subgoals(SubGoals)$
>> **else** $apply\_method(Mthd, G) = SubGoals$ **and** $refine\_subgoals(SubGoals)$
>
> **If** $\neg choose\_method(\_)$ **and** $critic\_applicable(G)$ **then**
>> $revise\_induction(G) = (NewScheme, Term)$
>> $store\_revision(NewScheme, Term)$
>> $move\_to\_nrst\_induction\_node$
>> $apply\_method(induction(NewScheme), G) = SubGoals$
>> $refine\_subgoals(SubGoals)$
>
> **else** FAIL

$map\_revision(Scheme, Term, G)$

Input: Goal G, induction scheme $Scheme$ and a term $Term$
Output: New induction scheme to apply on G

> $get\_maximal\_terms(G) = MaxList$.
> **If** $maps\_to(Term, MaxList) = Map$ **then**
>> $map\_scheme(Scheme, Map) = NewScheme$ **and return** $NewScheme$
>
> **else** FAIL

Fig. 1. Procedure for Planning with Critics and Internal Analogy

Justifications

Our internal analogy employs ideas from derivational analogy [Carbonell 86] that transfers a decision made in the source planning only if its justifications hold in the target. This means we have to store justifications of decisions which we are going to replay, in a justification slot of the plan nodes shown in Figure 2. On the one hand, the justifications should contain all information that is needed at

[8] To simplify matters, we have not included in the procedure any other critics given in [Ireland & Bundy 94], although these are present in CL^AM3.

```
PlanName:      the name of the conjecture being planned.
Address:       the address of the node within the plan tree.
Hypotheses:    the goal hypotheses.
Conclusion:    the goal conclusion.
Status:        the status of the goal sequent, open or closed.
Subst:         list of substitutions generated for meta variables.
Method:        the method chosen by the planner at that node.
Preconds:      successful and failed method preconditions tested at that node.
Count:         number of subgoals.
Justification: reasons for the decision made
```

Fig. 2. The Slots of a Proof Plan Node

replay time. On the other hand, the cost of capturing the justifications must be low, i.e., no complex computation can be performed.

The particular type of internal analogy we describe here can replay decisions about the induction scheme and induction variables provided by the induction revision critic. In Figure 1, the work of the critic is done by *revise_induction* which analyses the current sequent to produce a new induction scheme. *move_to_nrst_induction_node* then backtracks to the node N_1 of the plan at which the original induction scheme was applied. At this point, we store the following information in a justification slot of the node N_1:

- The revised induction scheme.
- The maximal subterm[9] of the induction hypothesis at N_1 which corresponds to the "blocked" term that motivated the critic application - i.e the term with which there is a partial wave rule match. Since this blocked term motivates the critic application and hence the revised induction term, the single maximal term associated with it, heuristically determines the result of the critic application.

This storing is done by *store_revision*, which creates a fact in the case base, $stored\_revision(Scheme, Term)$ to be accessed by the replay later. $Scheme$ is the revised induction scheme that was stored when applying the critic and $Term$ is the maximal subterm that was also stored.

As an example, refer again to the conjecture

$$\forall t, l : list(obj).\ even(length(t <> l)) = even(length(l <> t)) \qquad (6)$$

CIAM initially chooses a one-step induction. The step-case of this induction is planned until rippling becomes blocked, at which stage the current goal is:

[9] The maximal subterms of a term $f(t_1, t_2, ..., t_n)$ are the largest subterms of that term - i.e. $t_1, t_2, ..., t_n$

$$even(\boxed{s(\underline{length(t <> l)})}) = even(\boxed{s(\underline{length(l <> t)})}) \quad (12)$$

This is when the critic is applied. Hence, this node in the plan is our N_1, with induction hypothesis (6). In this case, there is a partial wave rule match between (8) and either the LHS or the RHS of (12). It does not matter which match is chosen - both lead to a successful induction revision. Suppose, the match between (8) and the RHS of (12) was chosen, then the justification for the critic application consists of:

- The revised induction term, $\boxed{h_1 :: h_2 :: \underline{l}}$ and the revised induction variable, $l : pnat\ list$[10].
- The maximal term of (6) corresponding to the blocked term that motivated the critic - namely, the RHS of (6), $even(length(l <> t))$.

The maximal subterm determines which induction scheme is required. For example, the maximal term $even(length(l <> t))$ tells us that a two-step list induction is needed. This is because $even$ determines a two-step induction and $length$ determines a list induction.

Mapping

The actual internal analogical replay proceeds by mapping and suggesting an induction scheme. At each induction node N_2, following the induction node N_1, we try to map the maximal term stored at N_1 to each of the maximal terms of the conclusion at N_2. This is done by $maps\_to(Term, MaxList)$ where $Term$ is the term at N_1 and $MaxList$ is the list of maximal terms of the conclusion at N_2. If at least one map is successful, we conjecture that an induction revision would take place following N_2, and we use internal analogy to suggest the revised induction scheme straight away, rather than waiting for the critic application.

After investigating several examples, it was discovered that the goals motivating the repeated critic calls tend to be very similar syntactically. Hence, only a very simple first-order mapping from the maximal term stored at N_1 to the maximal terms at N_2 was required. This means the procedure is low in computational cost.

Replay

Once the mapping has found that a critic application can be avoided, the replay suggests the revised induction scheme at N_2. This is done simply by applying this mapping to the induction term and the variable stored as part of the justification at N_1 (map_scheme takes care of this).

[10] *pnat list* is the type of lists of natural numbers

4 Example

To consolidate the ideas given in the previous section, we now present an extended example. Consider the conjecture

$$even(x + y) = even(y + x) \tag{16}$$

with the following available wave rules

$$even(\boxed{s(\boxed{s(\underline{U})})}) \Rightarrow even(U) \tag{17}$$

$$\boxed{s(\underline{U})} + V \Rightarrow \boxed{s(\underline{U + V})} \tag{18}$$

CI^AM finds a proof for (16) using the induction revision critic three times. The first revision comes straight away. CI^AM tries to carry out a one-step induction on y whereas in fact, a two-step induction is needed. The plan of the step-case produced by CI^AM up to the point of application of the first critic is as follows.

$$even(x + \boxed{s(\underline{y})}) = even(\boxed{s(\underline{y})} + x) \tag{19}$$

Applying (18)

$$even(x + \boxed{s(\underline{y})}) = even(\boxed{s(\underline{y + x})}) \tag{20}$$

Blocked since neither (17) nor (18) can be applied.

Now a critic is applied, which produces a revised term of $\boxed{s(s(\underline{y}))}$. The justification for this critic application consists of

1. The revised induction term $s(s(y))$
2. The variable to induce on, and its type $y : pnat$
3. The maximal term of (16) corresponding to the blocked term of (20) that motivates the critic application. Since there is a partial wave rule match between (17) and the RHS of (20), this maximal term is the RHS of (16), namely $even(y + x)$.

The planning process now continues as usual. During the planning of the step-case of this revised induction, CI^AM tries to plan the subgoal

$$even(a + s(s(b))) = even(a + b) \tag{21}$$

where the original variables have been replaced by fresh variables, a and b for reasons of clarity.

In the original proof, CI^AM3 now suggests a one-step induction on a and soon after makes another induction revision. Our analogy mechanism can be used to avoid this additional revision as follows.

Following the theory given in the previous section, try to obtain a map from the stored justification $even(y + x)$ to the maximal terms of (21). This gives two possible maps:

1. $even(y+x)$ to $even(a+s(s(b)))$ via $y \mapsto a, x \mapsto s(s(b))$
2. $even(y+x)$ to $even(a+b)$ via $y \mapsto a, x \mapsto b$

It does not matter which of these mappings we choose. In some cases, it may happen that a mapping maps the stored induction variable to a compound term. We disallow such mappings because they would produce an induction term different in structure to the stored term. For instance, if in the example, one mapping was $y \mapsto s(s(b))$, then this would map the stored induction term $\boxed{s(s(\underline{y}))}$ to $\boxed{s(s(s(s(\underline{b}))))}$, hence suggesting a four-step induction on b, which is clearly not required.

Returning to our example, taking either of our successful mappings, the stored induction term maps from $\boxed{s(s(\underline{y}))}$ to $\boxed{s(s(\underline{a}))}$ giving a two-step induction on a. Hence, the first subgoal in the step-case of (21) is

$$even(\boxed{s(\underline{\underline{\boxed{s(\underline{a})}}})} + s(s(b))) = even(\boxed{s(\underline{\underline{\boxed{s(\underline{a})}}})} + b) \tag{22}$$

The planning process continues on this subgoal as in the original proof.

Later in the proof, CL^AM3 needs to prove the subgoal

$$even(x+s(0)) = even(s(x)) \tag{23}$$

The maximal subterms (with new variable c) are $even(c+s(0))$ and $even(s(c))$. Mapping the stored $even(y+x)$ to the first of these subterms produces $y \mapsto c, x \mapsto s(0)$ (no map is possible from $even(y+x)$ to $even(s(c))$). The original revised induction term and variable, $\boxed{s(s(\underline{y}))}$ and $y : pnat$ are mapped to $\boxed{s(s(\underline{c}))}$ and $c : pnat$ respectively. This gives a two-step induction on c with conclusion

$$even(\boxed{s(\underline{\underline{\boxed{s(\underline{c})}}})} + s(0)) = even(s(\boxed{s(\underline{\underline{\boxed{s(\underline{c})}}})})) \tag{24}$$

Rippling can now take place as in the original proof.

In this example, analogy has been used to avoid two extraneous applications of the induction revision critic.

5 Implementation and Results

The procedure described above has been implemented as an extension to CL^AM3, a version of CL^AM equipped with critics. At present, CL^AM3 is very limited in its choice of induction schemes. It cannot cope, for example, with simultaneous inductions. As a result, the examples that the analogy procedure can be tested on are necessarily of a similar nature. However, we feel that if more diverse schemes were incorporated into CL^AM3, then our procedure is general enough to apply to the wider range of examples that these schemes would make available.

Sometimes, different types of induction revision can occur in the same proof. An example is

$$even(length(a <> b)) = even(length(b) + length(a)) \tag{25}$$

In this proof the first revision is from one-step list induction on a to two-step list induction on a, with induction term $\boxed{v0 :: v1 :: \underline{a}}$. The stored maximal term is $even(length(a <> b))$. The second critic is applied to the following goal which CL^AM3 suggests as a lemma and then tries to prove:

$$even(v0 + s(s(v1))) = even(v0 + v1) \tag{26}$$

resulting in a revised induction term of $\boxed{s(s(\underline{v0}))}$. Note that the two revisions suggest different induction terms. There is no map available from the stored justification here so an incorrect induction revision is *not* imposed, and the 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 and two induction revisions later in the proof can be eliminated so that the overall time is reduced.

Similarly, in the case that for a subsequent induction, only a one-step induction term is required (i.e. no critic is needed), no map from the stored justification will succeed, so the one-step induction is not altered.

It can also happen that once a revision has taken place, a mapping from the justification to a future goal is successful, even though CL^AM3 correctly chooses the induction scheme. An example occurs with the conjecture

$$half(x + y) = half(y + x) \tag{27}$$

The critic is applied almost immediately to suggest a two-step induction on y with induction term $\boxed{s(s(\underline{y}))}$. The justification stored is $half(y + x)$. Later in the original proof, the induction method is applied to the goal (suggested as a lemma)

$$half(x + 0) = half(x) \tag{28}$$

In this case, CL^AM3 would not invoke a critic application, but will suggest a two-step induction on x straight away because of the existence of the wave rule

$$half(\boxed{s(\boxed{s(\underline{\underline{U}})})}) \Rightarrow \boxed{s(\underline{half(U)})} \tag{29}$$

However, a mapping is available from the stored justification to the LHS of (28). This mapping is $\{y \mapsto x, x \mapsto 0\}$, which leads again to a two-step induction on x. So even though no critic application is avoided, the analogy can still be used to suggest an induction schema. 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.

The system was tested on the examples given in Table 1. The timings were taken on a Hewlett Packard 9000 715. Our system is written in Quintus Prolog v3.1.4. As expected, the costs associated 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.

Conjecture	T1	T2
$even(length(x <> y)) = even(length(y <> x))$	708	567
$half(length(x <> y)) = half(length(y <> x))$	329	295
$even(x + y) = even(y + x)$	48	40
$even(length(x <> y)) = even(length(y) + length(x))$	107	96
$half(x + y) = half(y + x)$	48	45
$even(length(x)) = even(length(rev(x)))$	141	122
$half(length(x)) = half(length(rev(x)))$	55	53
$even(z + length(x <> y)) = even(z + length(y <> x))$	3850	3502

T1 is the CPU time taken to plan the given theorem without using internal analogy. T2 is the time with our mechanism. All times are in seconds.

Table 1. Some Examples Run by our System

6 Conclusion

In general, it seems to be a good idea to introduce internal analogy into theorem proving if similar search-intensive subtasks can occur in a problem solving process. The reason is that in this case the retrieval and mapping efforts are small and the savings achieved by analogically replaying the result of a similar subtask are likely to more than compensate the additional effort for analogy.

While other papers on analogy in theorem proving such as [Owen 90] and [Kolbe & Walther 95] have lacked a discussion of why the application of analogy is advantageous in the situations their system faces, we substantiated the benefits of analogy in CI^AM3's situation.

In this paper we presented an internal analogy that has been used to reduce the number of critic applications required when planning some proofs. This analogy is done by carrying out the first induction revision on a goal G and looking for similarities between G and the current goal at each subsequent induction node, N. If there are sufficient similarities, the internal analogy conjectures that CI^AM3 is about to make use of a second induction revision, so it suggests an induction scheme at N which is analogous to that chosen at G's node. This avoids the further application of the critic. Some proof plans involved many such patches. As a result, the time taken to prove some theorems has been reduced.

For theorems that do not require an application of the induction revision critic the internal analogy is never invoked. For theorems with an application of the induction revision critic that cannot be replayed, an analysis has been delivered in section 5. There we have shown that the additional effort for analogy is small.

References

[Bundy *et al* 91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[Bundy *et al* 93] A. Bundy, Stevens A, F. Van Harmelen, A. Ireland, and A. Smaill. A heuristic for guiding inductive proofs. *Artificial Intelligence*, 63:185–253, 1993.

[Carbonell 86] 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.

[Gordon *et al* 79] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Lecture Notes in Computer Science 78. Springer, Berlin, 1979.

[Hickman & Larkin 90] Angela K. Hickman and Jill H. Larkin. Internal analogy: A model of transfer within problems. In *The 12th Annual Conference of The Cognitive Science Society*, pages 53–60, Hillsdale, NJ, 1990. Lawrence Erlbaum Associates.

[Kennedy 95] Angela K. Kennedy. Internal Analogy: A First Step Towards a Unified Theory of Analogical Problem Solving. Unpublished PhD thesis, Carnegie Mellon University, to appear 1995.

[Hutter 90] 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.

[Ireland & Bundy 94] 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.

[Kolbe & Walther 95] 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.

[Owen 90] S. Owen. *Analogy for automated reasoning.* Academic Press, London, 1990.

[Reif & Stenzel 93] W. Reif and K. Stenzel. Reuse of Proofs in Software Verification. In *Proc. 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1993. pages 74-85. LNCS 761, Springer.