Proof by analogy in mural

Vadera, S

http://dx.doi.org/10.1007/BF01211605

This is an author prepared version of:

S. Vadera, Proof by Analogy in Mural, Formal Aspects of Computing, Vol 7, No 9, pp183-206, 1995.

Proof by Analogy in mural

Sunil Vadera
Department of Mathematics and Computer Science,
University of Salford, Salford M5 4WT

1995

Abstract

An important advantages of using a formal method of developing software is that one can prove that development steps are correct with respect to their specification.

Conducting proofs by hand, however, can be time consuming to the extent that designers have to judge whether a proof of a particular obligation is worth conducting. Even if hand proofs are worth conducting, how do we know that they are correct?

One approach to overcoming this problem is to use an automatic theorem proving system to develop and check our proofs. However, in order to enable present day theorem provers to check proofs, one has to conduct them in much more detail than hand proofs. Carrying out more detailed proofs is of course more time consuming.

This paper describes the use of proof by analogy in an attempt to reduce the time spent on proofs. We develop and implement a proof follower based on analogy and present an example to illustrate its characteristics. The example shows that even when the follower fails to complete a proof, it can provide a hint that enables the user to complete a proof.

1 Introduction

Most theorem proving systems provide mechanisms by which a user can conduct proofs using rules in a forward (axiom driven) or backward (goal directed) manner, or even in mixed mode. As an aid to carrying out the proofs, some theorem provers provide a function that returns a list of possible candidate rules that match a goal. Although useful, if we use such a function at each possible proof step, then we simply end up with the explosive search space of resolution theorem provers. For example, in the theorem proving system *mural* [JJLM91], over 30 known rules match:

$$append(append(reverse(l), m), n) = append(reverse(l), append(m, n))$$

These include rules as diverse as the definition of set equality, induction rules and elimination rules.

We, of course, do not use such tools blindly. Instead, we use the experience gained in previous proofs to guide us. One way of embodying this experience in a theorem proving system is to develop explicit heuristics (e.g. Bundy's rippling tactics [Bun88]).

The development of explicit heuristics does, however, require considerable effort on our part. First we have to be able to recognise and generalise common proof steps. Then we have to code them in a tactic language. In the area of expert systems, even when the heuristics are known to human experts, the problem of generalising and representing them is known to be a major problem. Indeed, it is thought to be a bottleneck for the development of expert systems (e.g. see Buchanan et al. [B$^+$83]). It is therefore likely that the need to recognise and represent heuristics could form a bottleneck for their use in theorem proving.

An alternative to the explicit coding of heuristics is to carry out proofs by analogy. That is, one could use the proof of a similar lemma to aid the proof of a new lemma. If successful, such an approach would eliminate the need to explicitly code the heuristics and thereby reduce the impact of any bottleneck.

The use of analogy to aid the solution of problems is the subject of a broad range of studies. These include studies that examine psychological aspects of human problem solving processes (e.g. Gick and Holyoak [GH80]), attempts to use analogy to solve novel problems (e.g. McDermott [McD79]), and the use of analogy to aid automatic theorem provers. We refer the reader to Hall [Hal89] for a broad review of computational aspects of analogy.

In our use of analogy, we need to obtain a relationship that will enable us to use a source proof to obtain an analogous proof for the target lemma. Initially, we can relate the source and the target lemmas by finding a mapping from the source to the target lemma. We can then utilise this initial analogy to suggest proof steps for the target lemma from steps in the source proof. As we will see later, as the target proof develops, this initial analogy may be extended so that ultimately it relates the proofs as well as the lemmas.

This paper develops this notion of analogy further. Section 2 summarises the notation that is used. Section 3 develops our approach for carrying out proofs by analogy and section 4 gives an example to illustrate its characteristics. Section 5 concludes the paper and gives a brief comparison with other approaches to proof by analogy.

2 Summary of Notation

This section summarises the notation that is used in this paper. We have implemented our approach in *mural* and therefore adopt the notation and proof presentation style described in [JJLM91]. We also expect the reader to be familiar with logic and proofs as described in texts like [Bun79, Jon90, MW85].

Proof Presentation

A proof is presented as a sequence of lines. Assumption lines are labelled with a from and conclusion lines are labelled with an infer. Lines are justified on their right by giving the inference rule used and providing the lines that satisfy the hypotheses of the inference rule.

An inference rule takes the form:

$$\boxed{\text{Name}}\ \frac{\textit{List of Hypotheses}}{\textit{Conclusion}}$$

The hypotheses and the conclusion contain expressions that may include metavariables. The notation $E(x)$ denotes an expression E that is dependent on x. The hypotheses of an inference rule may include sequent hypotheses as well as ordinary hypotheses. For example, the sequence induction rule is written as:

$$\boxed{\text{Seq-Ind}}\ \frac{\begin{array}{c}P([\,]),\ s1\colon A^*,\\ [a, s]\{a\colon A, s\colon A^*, P(s) \vdash P(\text{cons}(a, s))\}\end{array}}{P(s1)}$$

where the first line consists of two ordinary hypotheses and the second consists of a sequent hypothesis that expresses the usual inductive step requirement. The list $[a, s]$ marks the variables bound by the sequent.[1] A sequent hypothesis is shown by carrying out a subproof. A subproof is shown in an indented fashion.

This paper presents several screen dumps of proofs carried out in the theorem proving system *mural*. Figure 1 presents an incomplete proof of the rule:

$$\boxed{\text{test source}}\ \frac{s1\colon A^*, s2\colon A^*}{\textbf{len}\,(s1 \frown s2) = \textbf{len}\,s1 + \textbf{len}\,s2}$$

In such screen dumps, hypotheses are labelled by '`h`' and conclusions by '`c`'. A *mural* justification consists of the rule used, references to the lines that satisfy the ordinary hypotheses, and reference to lines that satisfy the sequent hypotheses. An empty justification is denoted by `<Just>`.

The proofs that we carry out are in VDM's logic of partial functions. In particular, we use the notation $\delta(E)$ to mean that E is defined.

Instantiation

An instantiation records the mapping of metavariables to their expressions. The notation $E[\![e_i]\!]$ denotes an expression E that is dependent on $[\![e_i]\!]$, the i^{th} argument of E. For example, the conclusion of the proof in figure 1 is justified by the sequence induction rule with the following instantiation:

$$\{s1 \mapsto s1, P[\![e1]\!] \mapsto \textbf{len}\,([\![e1]\!] \frown s2)) = \textbf{len}\,[\![e1]\!] + \textbf{len}\,s2\}$$

[1] In mural, these are used to ensure the usual side condition that they do not occur free in any of the enclosing assumptions.

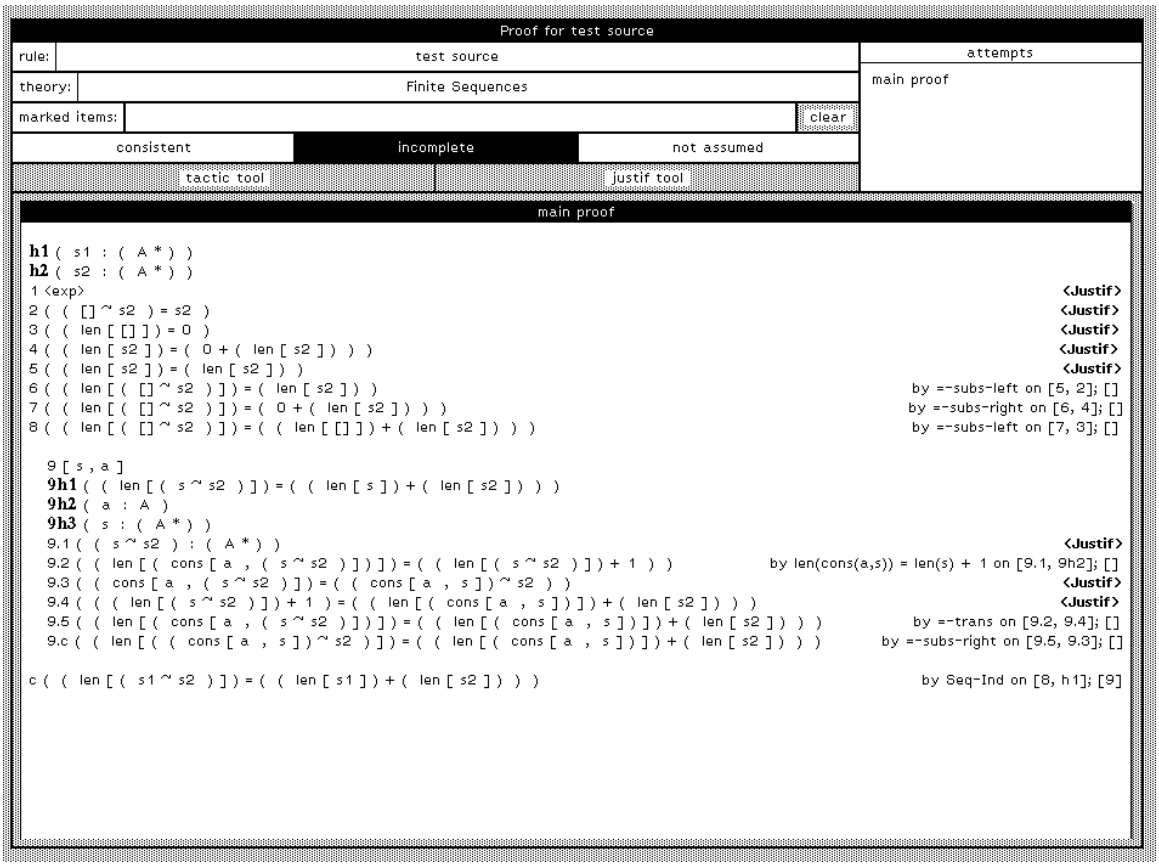

Figure 1: An (incomplete) proof in mural

3 A Proof Follower Based on Analogy

In this section we develop our approach to carrying out proofs by analogy. We first describe our basic approach in subsection 3.1. Then we show some weaknesses of this approach and refine it in subsection 3.2.

3.1 The Basic Approach

3.1.1 Obtaining an Initial Analogy

Given a theorem to prove, the first step in the use of analogy is to find an analogous lemma. Before we can find such a lemma we need to define what it means for a source lemma to be analogous to a target lemma.

Ideally, we would like to find a relationship that makes the source and target proofs analogous. However, initially, we only have the information contained in the source and target lemmas. Hence, we define a source lemma to be analogous to a target lemma if we can find a *symbol to symbol* mapping from the source lemma to the target lemma. This mapping, which we call an analogy, must be such that when it is applied to the source lemma, we obtain the target lemma. We will use the term matching to refer to the process of obtaining the analogy. Note that this process of matching differs from the kind of matching that makes two terms identical by the replacement of metavariables by terms. To avoid confusion, we will enclose the latter meaning of matching in quotes.
As an example, to prove:

$$\mathbf{elems}\,(a \frown b) = \mathbf{elems}\,(a) \cup \mathbf{elems}\,(b).$$

from a proof of:

$$\mathbf{card}\,(a \cup b) \leq \mathbf{card}\,(a) + \mathbf{card}\,(b).$$

we could obtain an initial analogy mapping:

$$\{\mathbf{card} \mapsto \mathbf{elems}\,, \cup \mapsto \frown, + \mapsto \cup,\ \leq \mapsto =\}$$

This analogy can then be used as a basis for guiding the target proof using the source proof. As the target proof develops, one can relate the target steps with the source steps and therefore extend the analogy so that it relates the proofs.

A Matcher

Informally, we can specify our matcher by requiring that after applying the analogy mapping to the source, it will become the target. We define the application of an analogy mapping as follows. Applying an analogy mapping θ to a term $f(a_1, a_2, ...a_n)$ gives:

1. $g(b_1, b_2, ...b_n)$ if f maps to g in θ, and the b_i are obtained by applying θ to the a_i.
2. $f(b_1, b_2, ...b_n)$ if f is not in the domain of θ, and the b_i are obtained by applying θ to the a_i.

An informal description of a matching algorithm which satisfies this specification is as follows. To match the terms $f(a_1, a_2, ...a_n)$ and $g(b_1, b_2, ..., b_n)$ in the context of an analogy θ we have the two cases:

1. f is in the domain of the current analogy θ. If f does not map to g in θ, then fail. Otherwise, repeatedly find an analogy θ_i from a_i to b_i in the context of $\theta_{i\text{-}1}$ with θ_0 being the existing analogy. If θ_n can be found successfully, then return θ_n, otherwise fail.

2. f is not in the domain of the current analogy θ. If f and g are not of the same class, (e.g. type symbols, variables, expression metavariables, type metavariables) then fail. Otherwise, repeatedly find an analogy θ_i from a_i to b_i in the context of $\theta_{i\text{-}1}$ with θ_0 being the current analogy with the additional mapping $\{f \to g\}$. If θ_n can be found successfully, then return θ_n, otherwise fail.

Note that the two terms do not match if they have different arities. A further point of interest is that as we build up the analogy, we also record the mapping of symbols to themselves. This is to ensure that we do not produce a mapping which does not satisfy our specification. For example, consider the source $f(h(x), h(d))$ and a target $g(h(x), k(d))$. If we do not record the mapping $\{h \mapsto h\}$, then we could obtain the mapping $\{f \mapsto g, h \mapsto k\}$ which, if applied to the source gives $g(k(x), k(d))$, which does not correspond to the target. For conciseness, we shall omit the presentation of identity mappings when they are not important.

3.1.2 Using and Extending the Analogy

Once we have obtained a mapping from the source to the target lemma, we need to carry out the proof using the analogy. There are at least two distinct ways in which we could carry out the proof:

1. We could work in a forwards manner; starting from known lines, and using these to justify new lines until we deduce the conclusion.

2. We could work in a backwards manner; starting from the conclusion, and generating subgoals. These subgoals can themselves be matched with lines in the source proof and followed in a backwards manner until all subgoals are proved. The process of matching subgoals with lines from the source proof may extend the initial analogy further.

For a human, the backwards approach is perhaps the most often used when faced with a new problem. However, the forwards approach is also useful in some situations. We prefer to provide a proof follower which proceeds in a goal directed, or backwards manner because:

- Not all the proof steps of the source can always be successfully mapped to the target. When such situations arise, the goal directed approach enables a user to continue the proof of such subgoals in a natural manner. With a forwards approach, one is not sure whether a line is an important subgoal or simply a subsidiary result

that helps in proving a subgoal which may be false in the target. That is, we believe that the backwards approach allows a human to pick up the loose threads of a proof more easily than a forwards approach.

- The important proof steps are encountered earlier if a backwards approach is taken. For example, the decision to use an induction rule is encountered towards the end of a forward proof, and at the beginning of a backwards proof.

Identifying a Rule

Before we can carry out a backward step, we need to identify an appropriate rule. We can base the identification of an appropriate rule on the rule used in the source proof. This is, in fact the major benefit of using analogy: it reduces our search space considerably. There are two situations that may arise:

1. The rule used in the source is the one that should be used in the target proof.
2. The rule used in the source is specific to the source proof and cannot simply be used in the target proof.

We describe each of these cases below.

Case 1: Source Rule can be used in the Target

The rule used in the source is the one that should be used in the target proof. Thus, for example, the source proof may contain the lines:

from ...
1 $\quad \mathbf{cons}(a, m) \frown n = \mathbf{cons}(a, m \frown n)$
2 $\quad \mathbf{len}\,(\mathbf{cons}(a, m) \frown n) = \mathbf{len}\,(\mathbf{cons}(a, m)) + \mathbf{len}\; n$
3 $\quad \mathbf{len}\,(\mathbf{cons}(a, m \frown n)) = \mathbf{len}\,(\mathbf{cons}(a, m)) + \mathbf{len}\; n$ $\quad$ =-subst-right(1,2)
infer ...

where

$$\boxed{\text{=-subst-right}}\ \frac{s1 = s2; E(s1)}{E(s2)}$$

and we would like to use it to obtain the target lines:

from ...
1 $\quad \mathbf{cons}(a, m) \frown n = \mathbf{cons}(a, m \frown n)$
2 $\quad \mathbf{elems}\,(\mathbf{cons}(a, m) \frown n) = \mathbf{elems}\,(\mathbf{cons}(a, m)) \cup \mathbf{elems}\; n$
3 $\quad \mathbf{elems}\,(\mathbf{cons}(a, m \frown n)) = \mathbf{elems}\,(\mathbf{cons}(a, m)) \cup \mathbf{elems}\; n$ $\quad$ =-subst-right(1,2)
infer ...

Such rules are not specific to the source theorem, and can therefore be used in the target. In general, we can identify such rules by noticing that applying the analogy mapping to

the rule has no effect on it. That is, the rule remains equivalent subject to the renaming of metavariables. Thus, in our example, applying the mapping $\{\mathbf{len} \mapsto \mathbf{elems}, + \mapsto \cup\}$ has no effect on the =-subst-right rule. There may be some occasions, for example when the analogy is not complete, when this heuristic suggests that the source rule should be used when a more specific one is appropriate.

Case 2: Source Rule cannot be used on the Target

The source proof may use rules that are specific to the source theorem, and which cannot simply be transferred to the proof of the target theorem. For example, consider the following lines of a source proof:

from ...
1 $s1$: X-**set**
2 $s2$: X-**set**
3 $s1 \cup s2 = s2 \cup s1$ $\cup$-comm(1,2)
infer ...

where

$$\boxed{\cup\text{-comm}}\ \frac{s1\text{: } X\textbf{-set}, s2\text{: } X\textbf{-set}}{s1 \cup s2 = s2 \cup s1}$$

and a target proof:

from ...
1 $s1$: X-**set**
2 $s2$: X-**set**
3 $s1 \cap s2 = s2 \cap s1$??
infer ...

We cannot, of course, simply use the $\cup$-comm rule in the target proof. It is specific to the source proof in that it is about $\cup$. However, we can apply the analogy that could be obtained by our matching algorithm to the $\cup$-comm rule to obtain an analogous rule:

$$\boxed{??}\ \frac{s1\text{: } X\textbf{-set}, s2\text{: } X\textbf{-set}}{s1 \cap s2 = s2 \cap s1}$$

The system can now scan the *current* theory to find this rule. It may be that the system can not find such a rule. In this case, the user could be prompted to add this, or a similar rule to the target theory. In such situations, the use of analogy to aid the proof of a target theorem results in 'new' rules for the target theory which themselves may need to be proved.

Notice that these two heuristics significantly improve upon the identification of rules by 'matching' a line with the conclusion of a rule.

The first case identifies the rule as the one used in the source and therefore performs no searching. Although the second case needs to scan the current theory, it is searching

for a specific rule which therefore reduces the number of potentially applicable rules.

Applying a Rule

Once we have identified a rule that might be appropriate for use in the target, we can apply it in a backwards manner. The manner in which we do this will affect the extent to which our analogy will succeed. Two of the extreme ways in which we may do this are:

- 'Match' the conclusion of the rule with the target line and add the instantiated hypotheses as subgoals.
- Use the source instantiation together with the analogy to obtain a target instantiation. This *analogous instantiation* can then be applied to the selected rule before adding the hypotheses as subgoals.

As an example to illustrate the difference, consider again the above example in which the =-subst-right rule was suggested by the source line. Then using the first approach, with 'matching', *mural* suggests some 18 possible instantiations. These include:

1. $\{s2 \mapsto \mathbf{elems}\ n\}$, giving:

   from ...
   1 $s1 = \mathbf{elems}\ n$
   2 $\mathbf{elems}\ (\mathbf{cons}(a, m \frown n)) = \mathbf{elems}\ (\mathbf{cons}(a, m)) \cup s1$
   3 $\mathbf{elems}\ (\mathbf{cons}(a, m \frown n)) = \mathbf{elems}\ (\mathbf{cons}(a, m)) \cup \mathbf{elems}\ n$=-subst-right(1,2)
   infer ...

2. $\{s2 \mapsto \mathbf{elems}\ (\mathbf{cons}(a, m)) \cup \mathbf{elems}\ n\}$, giving:

   from ...
   1 $s1 = \mathbf{elems}\ (\mathbf{cons}(a, m)) \cup \mathbf{elems}\ n$
   2 $\mathbf{elems}\ (\mathbf{cons}(a, m \frown n)) = s1$
   3 $\mathbf{elems}\ (\mathbf{cons}(a, m \frown n)) = \mathbf{elems}\ (\mathbf{cons}(a, m)) \cup \mathbf{elems}\ n$=-subst-right(1,2)
   infer ...

In each case, the metavariable $s1$ is left for the user to fill.

However, we can apply the analogy to the source instantiation:

$\{s1 \mapsto \mathbf{cons}(a, m) \frown n,$
$s2 \mapsto \mathbf{cons}(a, m \frown n),$
$E[\![e1]\!] \mapsto \mathbf{len}\ [\![e1]\!] = \mathbf{len}\ (\mathbf{cons}(a, m)) + n\}$

to obtain an analogous instantiation:

$\{s1 \mapsto \mathbf{cons}(a, m) \frown n,$
$s2 \mapsto \mathbf{cons}(a, m \frown n),$
$E[\![e1]\!] \mapsto \mathbf{elems}\,[\![e1]\!] = \mathbf{elems}\,(\mathbf{cons}(a, m)) \cup n\}$

which leads to the desired lines:

from ...
1 $\quad \mathbf{cons}(a, m) \frown n = \mathbf{cons}(a, m \frown n)$
2 $\quad \mathbf{elems}\,(\mathbf{cons}(a, m) \frown n) = \mathbf{elems}\,(\mathbf{cons}(a, m)) \cup \mathbf{elems}\, n$
3 $\quad \mathbf{elems}\,(\mathbf{cons}(a, m \frown n)) = \mathbf{elems}\,(\mathbf{cons}(a, m)) \cup \mathbf{elems}\, n$ $\quad$ =-subst-right(1,2)
infer ...

Thus, as this example illustrates, analogy is not only useful when selecting a rule, but also in obtaining a suitable instantiation of a rule.

3.1.3 Summary of the Basic Approach

We have now described the rationale behind our approach and can summarise it as follows. Given a goal *Gs* in a source proof, and a target goal *Gt*:

1. Find an analogy θ from *Gs* to *Gt*. If no analogy exists then fail.
2. If an analogy θ exists, then use the rule used to justify *Gs* to obtain a rule that could justify *Gt* as follows:
   - If the rule used to justify *Gs* is general (e.g. =-trans, =-subt-left...), then use it to justify *Gt*. This is detected by noting that in such circumstances, applying the analogy θ has no effect on the rule.
   - If the rule used to justify *Gs* is more specific, then use the analogy θ to try and obtain an analogous rule. Search the *current* theory for an equivalent rule (subject to the renaming of metavariables) in order to identify the rule.

     If this fails, display the analogous rule and allow the user to identify the rule, add the rule, or abandon the current goal.
3. If a rule is found in the preceding step, use the source instantiation to obtain a target instantiation, and apply the instantiated rule in a backwards fashion.

   The backwards application of a rule is achieved by simply adding those hypotheses that are not already in the proof.

   The resulting lines in the target may have analogous lines in the source. Hence, use matching to associate any new lines with those source lines referenced by the source justification. This may result in an extension to the analogy. The process can now be repeated (from step 2) on the associated lines in a depth first manner.
4. If no rule can be found to justify *Gt*, then leave it for the user to prove.

In subsequent sections we refer to the approach that we develop as the *proof follower*.

3.1.4 The Benefit of Strict Matching

Our view of analogy as matching is quite strict. We could consider two lemmas to be analogous even if their terms have different arities. We could also be looser by allowing matching between arguments in different positions of the two lemmas.

However, a major benefit of this strict approach is that we can have some confidence in using analogous instantiations. In particular, if we adopt this level of strictness, we have the following result (whose proof can be found in [Vad95]).

Theorem: The Benefit of Strict Matching

Given:

- an analogy θ from a source line Gs to a target line Gt,
- a source instantiation Is, such that instantiating the source rule Rs produces Gs as conclusion,
- that θ applied to Rs gives an analogous rule Rt,
- a target instantiation It that is obtained by applying the analogy to the source instantiation Is

then, instantiating the rule Rt by It gives a conclusion that is equivalent to the target line Gt.

That is, provided the conditions are met, and we adopt the strict notion of analogy, an analogous instantiation will ensure that the analogous rule's conclusion is equivalent to the target line.

This lemma is not meant to suggest that a strict notion of analogy is necessarily the best. Instead, it offers a firm starting point for the use of analogous instantiations. The further we depart from it, the more work our follower will have to do to obtain and use analogous instantiations and new proof lines. If we depart from it too far, then we may be unable to use analogous instantiations and end up carrying out an unguided backwards proof. On the other hand, if we adopt it too strictly, we may find that our follower is useful in only a few situations.

3.2 Refinement of the Basic Approach

As described so far, our proof follower is not particularly successful in doing proofs by analogy. For example, given our earlier partial proof (figure 1) of:

$$\boxed{\text{test source}}\ \frac{s1\colon A^*, s2\colon A^*}{\mathbf{len}\,(s1 \frown s2) = \mathbf{len}\,s1 + \mathbf{len}\,s2}$$

it produces the proof given in figure 2 for the following target lemma:

$$\boxed{\text{test target}}\ \frac{s1\colon A^*, s2\colon A^*}{\mathbf{elems}\,(s1 \frown s2) = \mathbf{elems}\,s1 \cup \mathbf{elems}\,s2}$$

Although not entirely hopeless, it does leave the user to correct a number of lines (i.e. lines 4,5,6, 8.1, 8.2). For instance, the zero on line six needs replacing by the empty set.

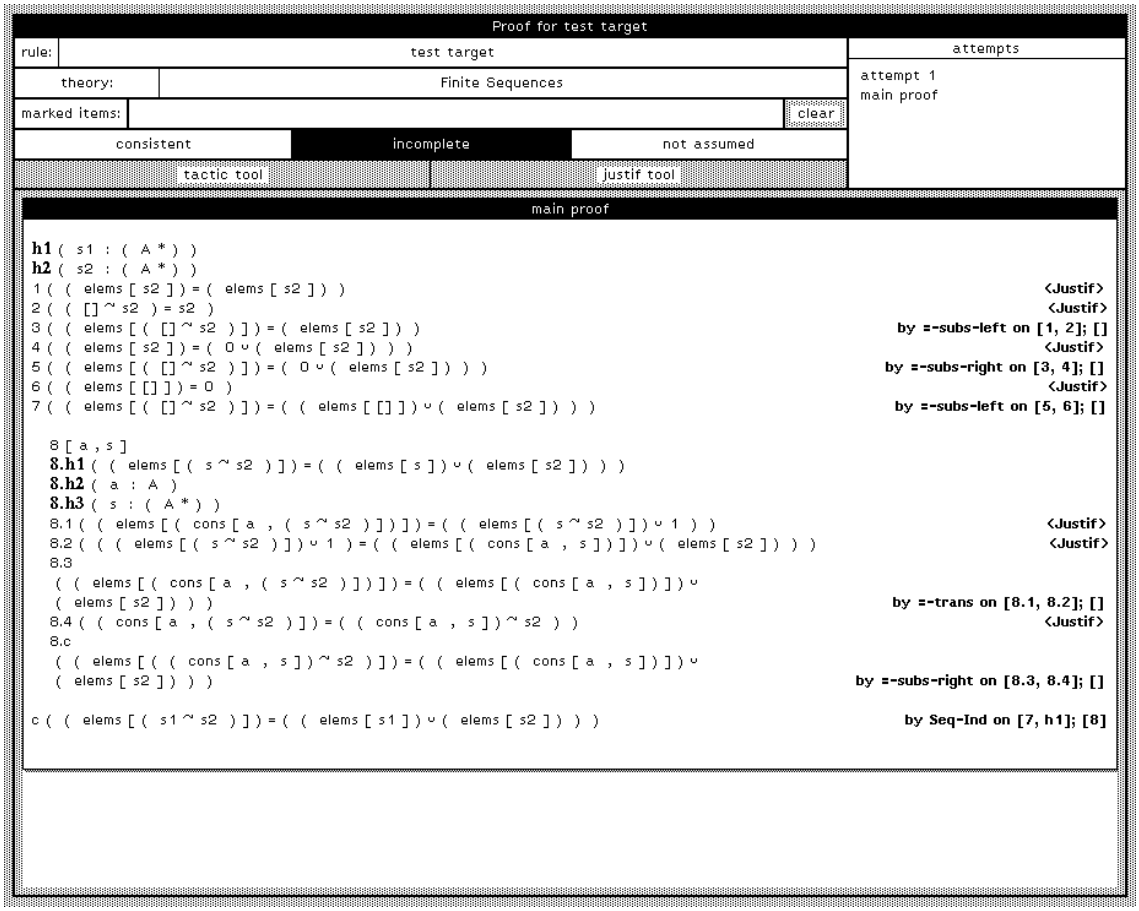

Figure 2: Follower's attempted proof

Of course, making these corrections requires less effort than doing the whole proof. However, these errors are typical of this scheme and are likely to occur again and again. This section therefore illustrates various problems that can arise and describes our solution to them.

3.2.1 Compensating for Incomplete Analogies

Most of the errors in figure 2 occur because the analogy used is incomplete. Initially, the analogy is obtained by matching a source lemma to a target lemma. This analogy is then used to obtain an analogous rule, and an analogous instantiation. The analogous rule and instantiation are then used in a backwards manner, as described earlier, to generate new target lines. Subsequently, the analogy may be extended by matching a new target line

to a source line. At any stage in this process, there is no guarantee that the analogy is complete. Although our theorem guarantees that the analogous rule's conclusion matches the target, there is no guarantee that the resulting new lines are correct.

Below, we illustrate the problems that arise due to an incomplete analogy and our solution to them.

Wrong Constant Problem

Consider the following source lines:

from ..
9.2 $\quad \mathbf{len}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{len}\,(s \frown s2) + 1 \qquad$ len(cons(a,s)) =...
9.4 $\quad \mathbf{len}\,(s \frown s2) + 1 = \mathbf{len}\,(\mathbf{cons}(a, s)) + \mathbf{len}\; s2$
9.5 $\quad \mathbf{len}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{len}\,(\mathbf{cons}(a, s)) + \mathbf{len}\; s2 \qquad$ =-trans(9.2,9.4)
infer ...

where the =-trans rule is defined as:

$$\boxed{\text{=-trans}}\ \frac{X = Y, Y = Z}{X = Z}$$

and the instantiation used in the justification for line 9.5 is:

$\{X \mapsto \mathbf{len}\,(\mathbf{cons}(a, s \frown s2)),$
$\;\; Y \mapsto \mathbf{len}\,(s \frown s2) + 1,$
$\;\; Z \mapsto \mathbf{len}\,(\mathbf{cons}(a, s)) + \mathbf{len}\; s2\}$

and a target line (8.3):

from ...
8.3 $\quad \mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\; s2$
infer ...

Then matching 9.5 to 8.3, we have an analogy:

$$\{\mathbf{len} \mapsto \mathbf{elems}\,, + \mapsto \cup\}$$

Since this analogy does not contain an analogue for 1, it results in an analogous instantiation:

$\{X \mapsto \mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)),$
$\;\; Y \mapsto \mathbf{elems}\,(s \frown s2) \cup 1,$
$\;\; Z \mapsto \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\; s2\}$

which in turn results in the following lines:

Table 1: Table of built in analogies

Expression	Built-in Analogy
$\{\}: X$**-set**	$0: \mathbf{N}$
$[]: X^*$	$0: \mathbf{N}$
$[]: X^*$	$\{\}: X$**-set**
$\{M\}: X$**-set**	$1: \mathbf{N}$
$[M]: X^*$	$1: \mathbf{N}$

where M is a metavariable that would need to be filled (a process that we illustrate below).

from ...
8.1 $\quad \mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{elems}\,(s \frown s2) \cup 1$
8.2 $\quad \mathbf{elems}\,(s \frown s2) \cup 1 = \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\; s2$
8.3 $\quad \mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\; s2$ $\quad$ =-trans(8.1,8.2)
infer ...

where lines 8.1 and 8.2 are not quite correct. We solve this kind of problem by building in the analogies between those constants that have similar algebraic properties (as given in table 1).

This pragmatic solution does, unfortunately, take a small step away from our ideal of automatically finding the complete analogy. However, it is a price worth paying for avoiding these kinds of errors.

It is worth noting that this is not the only set of possible analogies. For example, one could associate (1,*) in **N** with (S,∩) in sets. Our choice of associations is based on constructors since these are the most likely analogies that we would need in proving properties of programs.

Clearly we cannot simply substitute all occurrences of one constant by its analogue. For example in:

$$\mathbf{card}\,\{\} + \{\}$$

we only need to correct the second occurrence of $\{\}$.

Hence, before we can use these corrections, we need to detect when, and where to use them. We can detect most situations by checking the type of the expression. Thus, in the above example, we note that the second argument of + is not a number. We can do this provided that the signatures of the functions involved are given by the user.

Then such cases are corrected by the following algorithm. Given an expression $f(a_1, ..., a_n)$ which we expect to have a type T:

1. Find a signature rule of the form:

$$\boxed{\text{SigRule}}\ \frac{m_1{:}\ Tm_1, m_2{:}\ Tm_2, ..., m_n{:}\ Tm_n}{f(m_1, m_2, ...m_n){:}\ Tm}$$

2. If $Tm \neq T$, then since we expect f to be of type T, we can use the table to find a

replacement g which is of type T by indexing the table with f: Tm. We then check that $g(m_1, ...m_n)$: T.

3. If $Tm = T$, then check that: m_1: Tm_1, m_2: Tm_2, ..., m_n: Tm_n, and let the returned expressions be: $e_1, ...e_n$. Then return $f(e_1, ...e_n)$.

We add two special cases to this scheme. First, when an expression to be checked is a type requirement E: T and the type of E is Tm which is not T, then we are not sure whether to correct E or T. In general, our initial analogy does not contain much information about types. Hence, in such circumstances we return E: Tm.

Second, this scheme does not cope with equality. For example, given a situation:

$$f(...) = g(...)$$

where the range of f and g are not compatible, we do not know which one to correct.

In general, one can expect that most of the function symbols involved in the analogy are present in the lemmas. However, this is less likely for constants. Thus, our analogies are more likely to omit information about constants than about the mapping of functions. Hence, in our scheme, the most common occurrence of this kind of type mismatch is when either f or g may be a constant. For example, the following line occurs in figure 2:

$$\mathbf{elems}\,[\,] = 0$$

We therefore extend our scheme to detect such cases and correct the constant. Thus, for the above lines (bottom of page 13), we would now obtain the following lines:

from ...
8.1 $\mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{elems}\,(s \frown s2) \cup \{M\}$
8.2 $\mathbf{elems}\,(s \frown s2) \cup \{M\} = \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\,s2$
8.3 $\mathbf{elems}\,(\mathbf{cons}(a, s \frown s2)) = \mathbf{elems}\,(\mathbf{cons}(a, s)) \cup \mathbf{elems}\,s2$ =-trans(8.1,8.2)
infer ...

where M is a metavariable that still needs to be filled.

As a consequence of this kind of correction the analogous rule for line 8.1 becomes:

$$\boxed{??}\ \frac{a\colon X, s2\colon X^*}{\mathbf{elems}\,\mathbf{cons}(a, s2) = \mathbf{elems}\,s2 \cup \{M\}}$$

Hence the rule identification procedure that we described above would fail to identify the following rule (even if it is present):

$$\boxed{\text{elems-defn}}\ \frac{a\colon X, s2\colon X^*}{\mathbf{elems}\,\mathbf{cons}(a, s2) = \mathbf{elems}\,s2 \cup \{a\}}$$

We therefore extend our rule identification procedure so that in cases like this, where we have introduced a free metavariable, we look for a rule whose conclusion matches the proposed rule. This enables us to find a filler for the free metavariable which is then instantiated throughout the proof. In the above example, the rule and the filler for M would be detected on line 8.1 and would lead to the correction of lines 8.1 and 8.2.

Wrong Type in the Hypotheses of a Rule

Another problem that results from the lack of a complete analogy is illustrated by the following situation. Consider the source lines:

from ...
9.1 **len** $s1$: **N**
9.2 **len** $s2$: **N**
9.3 **len** $s1$ + **len** $s2$ = **len** $s2$ + **len** $s1$ +- comm(9.1,9.2)
infer ...

and a target line:

from ...
10.3 **elems** $s1$ ∪ **elems** $s2$ = **elems** $s2$ ∪ **elems** $s1$
infer ...

Our matcher will give an analogous mapping:

$$\{\textbf{len} \mapsto \textbf{elems}\,, + \mapsto \cup\}$$

This however, results in an analogous rule:

$$\boxed{??}\ \frac{s1\colon \mathbf{N},\ s2\colon \mathbf{N}}{s1 \cup s2 = s2 \cup s1}$$

If this is displayed, the user would probably identify the ∪-comm rule as the one that should be used. But one cannot help feeling disappointed that this rule is so close, and yet needs help from a user.

Our solution to this problem is based on correcting the types. This time we use the information in the conclusion to deduce the types, and then feed that information back into the hypotheses. Thus in this case, the consequent tells us that $s1$ and $s2$ must be sets, and this is used to correct the hypothesis.

Once this correction is performed, our system can find the ∪-comm rule by searching for a rule that is equivalent[2] to the corrected analogous rule.

In addition to using the signature rules to aid the above corrections, our refined approach also uses them to justify proof lines that are type requirements (where appropriate).

3.2.2 Loosening the Basic Approach

So far, our illustrative examples have adhered to the strict notion of analogy that we have adopted. We now present two refinements that loosen this notion a little.

[2]Subject to the renaming of metavariables.

Loosening the Matcher

The following example illustrates the need to loosen the matcher a little. Given a source proof:

from ...
9.1 $\{a\}$: X-**set**
9.2 **elems** $s2$: X-**set**
9.3 $\{a\} \cup$ **elems** $s2 =$ **elems** $s2 \cup \{a\}$ $\cup$-comm(9.1,9.2)
infer ...

and a target:

from ...
10.3 $1 +$ **len** $s2 =$ **len** $s2 + 1$
infer ...

Then, clearly we would like 10.3 to be analogous to 9.3. Unfortunately, the arities of $\{a\}$ and 1 are different. We therefore loosen our matching so that it *ignores* differences like this one. Hence, we obtain an analogy:

$$\{\textbf{elems} \mapsto \textbf{len}\,, \cup \mapsto +\}$$

When we apply this analogy to the $\cup$-comm rule we obtain:

$$\boxed{??}\ \frac{s1\text{: }X\textbf{-set}, s2\text{: }X\textbf{-set}}{s1 + s2 = s2 + s1}$$

After applying our type correction we obtain a rule:

$$\boxed{??}\ \frac{s1\text{: }\mathbf{N}, s2\text{: }\mathbf{N}}{s1 + s2 = s2 + s1}$$

This enables us to identify the +-comm rule, but since our instantiation is incorrect:

$$\{s1 \mapsto \{a\}, s2 \mapsto \textbf{len}\ s2\}$$

we would obtain an instantiated rule:

$$\boxed{\text{+-comm inst}}\ \frac{\{a\}\text{: }\mathbf{N}, \textbf{len}\ s2\text{: }\mathbf{N}}{\{a\} + \textbf{len}\ s2 = \textbf{len}\ s2 + \{a\}}$$

Hence, we also use the information about the types which is obtained from the conclusion, to check and correct the instantiation. So that in this case, our instantiation becomes:

$$\{s1 \mapsto 1, s2 \mapsto \textbf{len}\ s2\}$$

This then enables us to use the instantiated rule in a backwards fashion to obtain the desired lines:

from ...
10.1 $1 : \mathbf{N}$
10.2 $\mathbf{len}\ s2 : \mathbf{N}$
10.3 $1 + \mathbf{len}\ s2 = \mathbf{len}\ s2 + 1$ +-comm(10.1,10.2)
infer ...

Loosening the Rule Identification Procedure

In our basic approach, when the analogy changes the source rule, we look for a target rule that is equivalent to the proposed analogous rule. When this fails, we leave it to the user to identify the rule. However, there are circumstances in which the conclusions are equivalent, but the hypotheses differ slightly. For example, the following two rules are similar:

$$\boxed{\text{len(cons(a,s)) ...}}\ \frac{s : X^*,\ a : X}{\mathbf{len}\ (\mathbf{cons}(a, s)) = \mathbf{len}\ (s) + 1}$$

$$\boxed{\text{card-def-}\neg\in}\ \frac{s : X\textbf{-set},\ a : X,\ \neg(a \in s)}{\mathbf{card}\ (\text{add}(a, s)) = \mathbf{card}\ (s) + 1}$$

although their hypotheses differ slightly. Hence, when we fail to find a rule that is equivalent to the proposed rule, we relax our requirements and look for a rule whose conclusions are equivalent but whose hypotheses may differ. If this still fails, then we let the user identify the rule, add it, or abandon the current goal.

3.2.3 Renaming an Instantiation

When identifying a rule using a proposed rule, we would obviously like to accept a target rule if it only differs in the naming of its metavariables. Hence, in our description above, we have qualified that we are looking for a rule that is equivalent subject to the renaming of metavariables. For example, if the proposed analogous rule is:

$$\boxed{??}\ \frac{s1 : \mathbf{N},\ s2 : \mathbf{N}}{s1 + s2 = s2 + s1}$$

our rule identification procedure would identify the following target rule which only differs in the naming of its metavariables:

$$\boxed{+\text{-comm}}\ \frac{n1 : \mathbf{N},\ n2 : \mathbf{N}}{n1 + n2 = n2 + n1}$$

However, this means that we have to rename the analogous instantiation. For example, earlier we had the analogous instantiation:

$$\{s1 \mapsto 1, s2 \mapsto \mathbf{len}\ s2\}$$

Since the actual rule is in terms of $n1$ and $n2$, we need to rename the domain of the instantiation so that $s1$ becomes $n1$ and $s2$ becomes $n2$. In general, we use matching

to obtain a renaming from the proposed analogous rule to the actual rule and then use it to rename the domain of the instantiation. The metavariables in the range are not renamed since they refer to the metavariables in the proof. They may of course have been renamed by the analogy to ensure that they refer to metavariables in the target proof instead of the source proof.

3.2.4 Selecting an Induction Rule

The lack of a complete analogy can also create problems in selecting an appropriate induction rule. For example, the following source:

$$\mathbf{card}\,(a \cup b) \leq \mathbf{card}\,a + \mathbf{card}\,b$$

and the target:

$$\mathbf{elems}\,(a \frown b) = \mathbf{elems}\,a \cup \mathbf{elems}\,b$$

would give an analogy mapping:

$$\{\mathbf{card} \mapsto \mathbf{elems}\,, \cup \mapsto \frown,\ \leq \mapsto =, + \mapsto \cup\}$$

Applying this mapping to the set induction rule:

$$\text{Set-Induction-Up}\ \frac{[a, s]\{a\colon A, s\colon A\textbf{-set}, P(s), \neg(a \in s) \vdash P(\mathrm{add}(a, s))\},\ P(\{\,\}), s1\colon A\textbf{-set}}{P(s1)}$$

does not change it. Hence, as it stands, our follower would incorrectly attempt to use this rule. In this example, the problem would be avoided if it could have managed to discover that add is analogous to cons. We therefore extend our rule identification procedure with the rule:

> If the source rule is an induction rule and the source and target theory are different then let the user select the analogous induction rule.

A more sophisticated approach is possible. We could determine the type of the variable over which the induction should be carried out, and then attempt to select an appropriate induction rule. Thus, in this case we would determine that the induction variable is of type sequence, and therefore induction over sequences should be carried out. We have, however, only implemented the simpler approach.

4 An Example use of the Proof Follower

We have implemented the proof follower in the theorem proving system *mural* and have carried out several examples. It can prove the following lemma (on which the basic approach failed):

$$\text{elems..}\ \frac{s1\colon T^*, s2\colon T^*}{\mathbf{elems}\,(s1 \frown s2) = \mathbf{elems}\,s1 \cup \mathbf{elems}\,s2}$$

from a source proof of:

$$\boxed{\text{len..}}\ \frac{s1\text{: } T^*,\ s2\text{: } T^*}{\mathbf{len}\,(s1 \frown s2) = \mathbf{len}\,s1 + \mathbf{len}\,s2}$$

These proofs can be found in [Vad95]. The reader can also refer to [Vad92] for more involved examples. Here, we concentrate on an example that illustrates some characteristics of the proof follower.

One way of proving the associativity of set union is to use its definition in terms of disjunction and then to use the associative property of disjunction. The proof follower is able to produce a complete proof of the associativity of set intersection from such a source proof.

An alternative way of proving the associativity of set union is to use induction. Thus, for example, Jones [Jon90] adopts this approach. We therefore carried out the proof of $\cup$-assoc by induction in *mural* as shown in figure 3.[1] How well does the proof follower manage to carry out the proof of $\cap$-assoc from this inductive proof? Figure 4 shows the result of its attempt.

The proof lines in the base case are wrong. In attempting to justify line 5 by using line 2 in the source proof, it proposes the following (incorrect) analogous rule to the user:

$$\boxed{??}\ \frac{s\text{: } A\textbf{-set}}{\{\,\} \cap s = s}$$

The user, of course, selects the right rule ($\{\,\} \cap s = \{\,\}$). However, although the proof follower records the selected rule in the justification of line 5, it does not correct the expression on line 5. A similar problem also occurs on line 3 of the target proof.

The induction step is incomplete. There are three lines in the induction step that still need justifying. These three lines take the form:

$$\begin{array}{ll} 7.5 & a \in s2 \\ 7.9 & a \in s3 \\ 7.13 & a \in (s2 \cap s3) \end{array}$$

Since the follower proceeds in a goal driven fashion, it should be clear that the follower's attempt to prove the induction step (box 7) is dependent on these three lines. But we cannot assume that a is a member of $s2$ and $s3$. Hence, this should trigger a thought that the proof should be done by cases on:

$$a \in (s2 \cap s3) \vee \neg(a \in (s2 \cap s3))$$

It should also be apparent that the follower is attempting to prove the case when $a \in (s2 \cap s3)$. Figures 5, and 6 show the completed proof by cases.

With the aid of copying and pasting, most of the proof lines and justifications in the case when $a \in (s1 \cap s2)$ are provided by the follower (box 8.7, figure 6). The case when $\neg(a \in (s1 \cap s2))$ is carried out without any aid from our follower (box 8.8, figure 6).

[1] The reader should be able to deduce the content of the rules used from the justifications. The reader can also refer to [Vad92] for the rules.

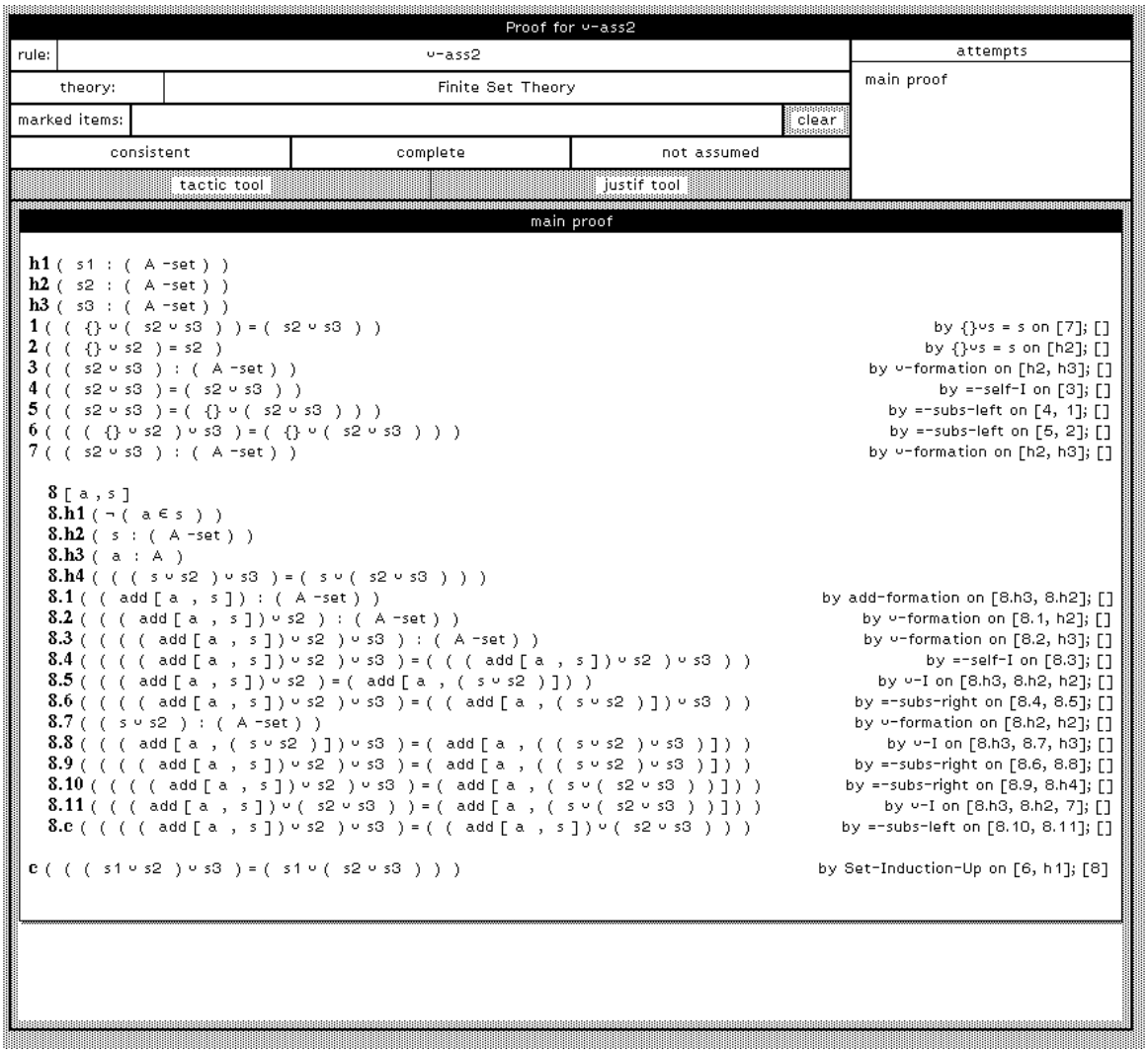

Figure 3: Source proof of ∪-assoc (inductive)

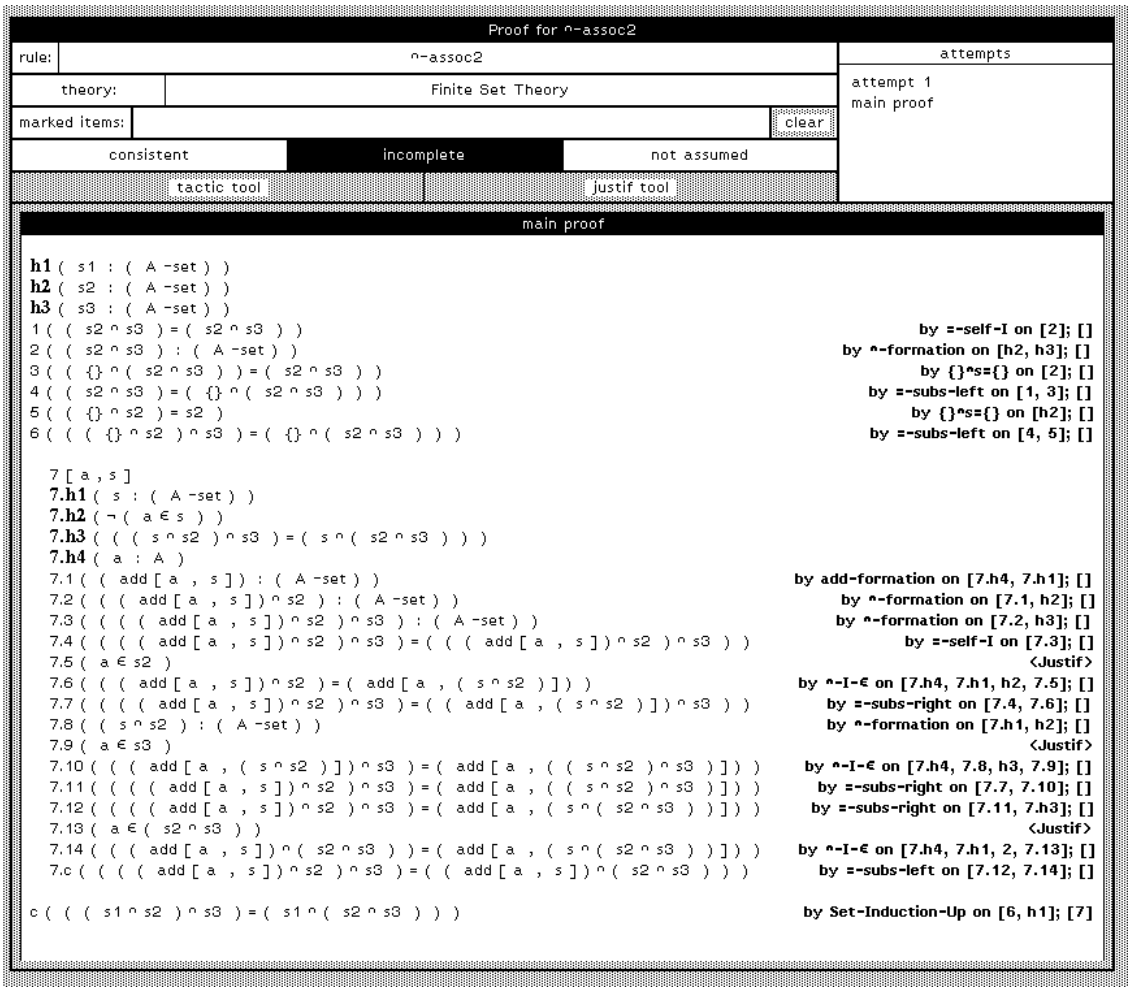

Figure 4: Follower's proof of ∩-assoc (inductive)

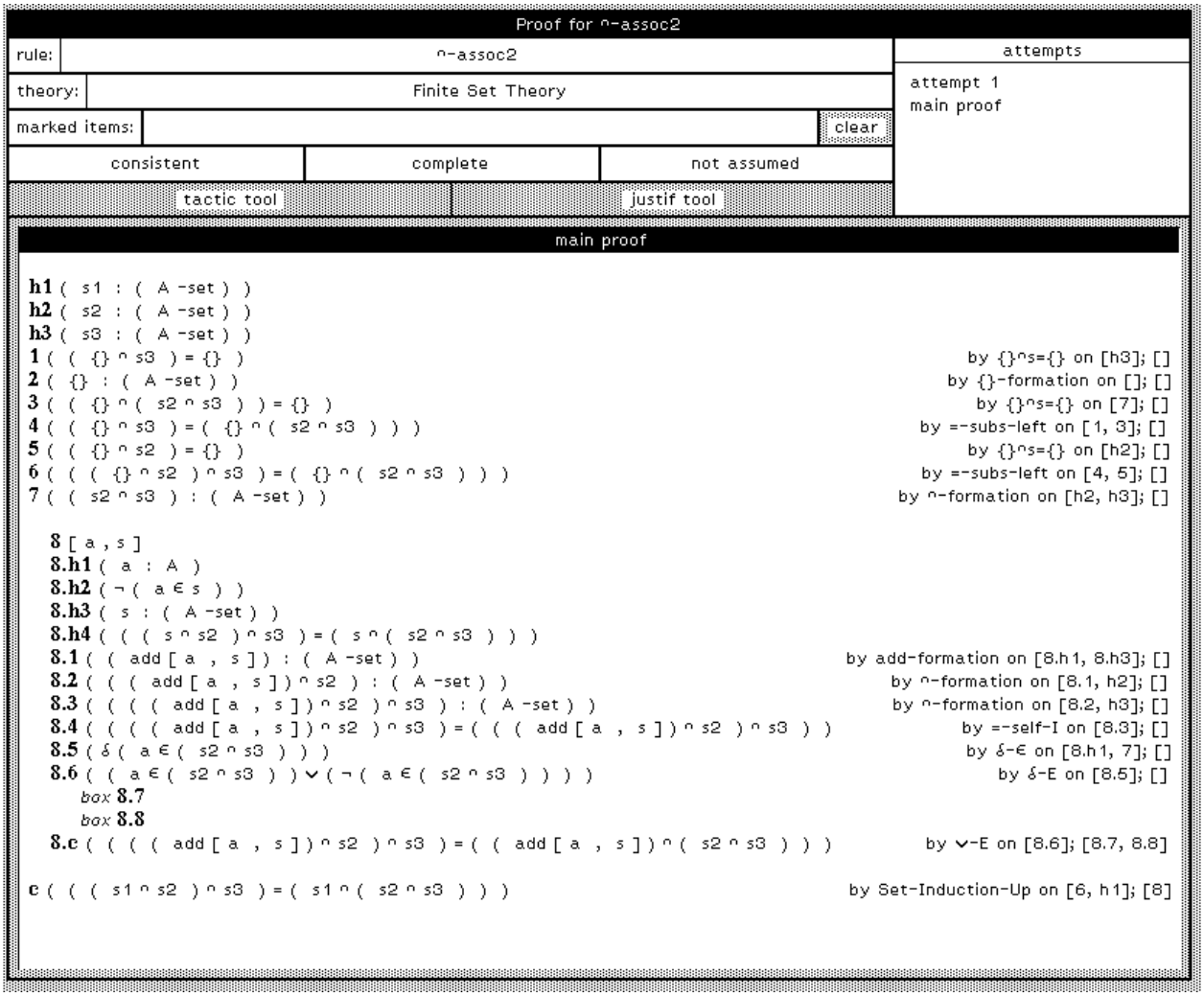

Figure 5: Top level of the complete proof of ∩-assoc

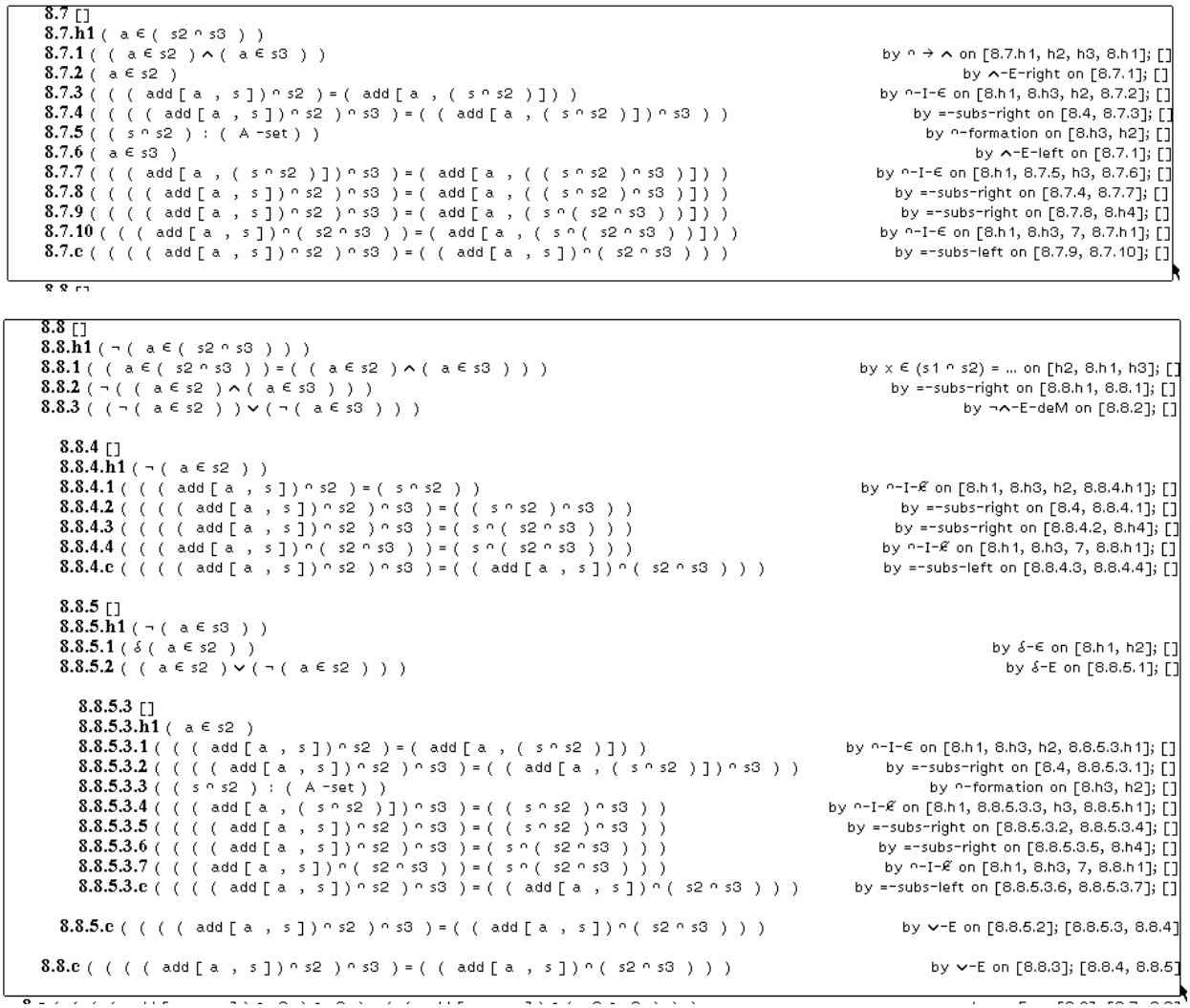

Figure 6: Boxes 8.7 and 8.8 of the complete proof of ∩-assoc

As for the base case (figure 5), once again things are not as bad as they first appeared. The only modification that is needed is to replace $s2$, and $s2 \cap s3$ by $\{\,\}$ in appropriate places, use the $\{\,\} \cap s = \{\,\}$ rule in line 1 instead of =-self-I, and to add line 2.

Hence, although our follower has not managed to complete the proof of associativity of intersection from the inductive proof of the associativity of set union, it has made a significant contribution. Out of the 46 lines of the completed proof (excluding hypotheses) 28 are new lines, 4 are modified versions of those produced by the follower, and 14 are provided by the follower. This example also illustrates that the success of the follower depends on the way a source proof is carried out as well as the source lemma involved. It should not, however, be taken to mean that the follower is less successful when inductive proofs are used as sources.

5 Discussion

We have developed a proof follower that is capable of carrying out proofs by analogy. It matches a given source lemma to a target lemma to obtain an initial analogy. This analogy is then used to carry out a backwards proof of the target lemma by analogy with the source proof. As this backwards proof by analogy proceeds, the initial analogy may be elaborated as new target lines are matched with source lines.

In comparison to an unguided backwards proof, this approach reduces the amount of searching in two ways. First, rules are identified by looking for an equivalent analogous rule and not by higher order matching. The use of higher order matching can result in many irrelevant rules but one can expect only a small number of equivalent analogous rules. Second, the approach uses the instantiation used for a source rule to suggest an analogous instantiation for the identified target rule. The use of higher order matching would propose several alternative instantiations at each backward step. An unguided backwards proof can therefore result in an explosive search space. In contrast, the proof follower implemented does not employ a tentative search strategy at all. Of course, this does sacrifice completeness and does assume that a suitable source proof is available. That is, there are examples for which a proof is possible for which the proof follower will fail.

There have been several other attempts at carrying out proofs by analogy.[2] The most notable contributions are the resolution based approaches of Kling [Kli71], Munyer [Mun81], and Owen [Owe90]. A primary aim of these systems is to reduce the large number of axioms that are available as potential parent clauses in a resolution proof. Hence these systems use a source lemma and the axioms used in its proof as a basis for filtering axioms that could be relevant in a proof of the target lemma (as summarised by figure 7).

Given that it is difficult to continue failed resolution proofs, these approaches have adopted a much looser notion of matching so that relevant axioms are not excluded. A primary benefit of the looser matchers is that a broader range of analogies can be

[2] For conciseness, we discuss the main points here but refer the reader to [Vad95] for more discussion.

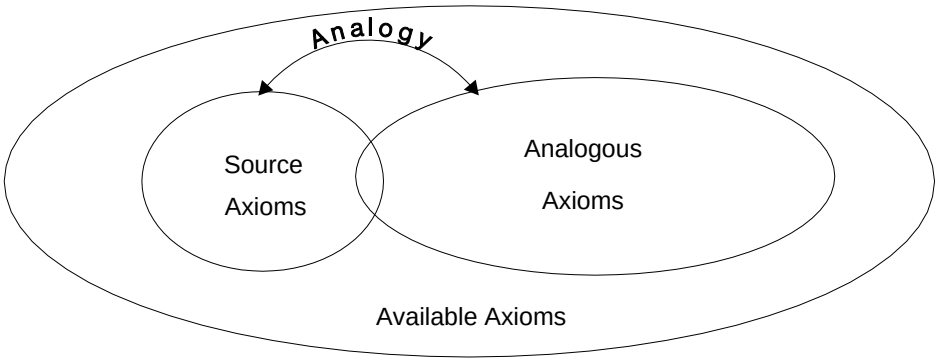

Figure 7: Filtering Axioms by Analogy

detected. For instance, Owen's matcher can detect an analogy between:

$$s1 \cap s2 \subseteq s1$$

$$s1 \subseteq s1 \cup s2$$

However, this advantage is gained at the following cost:

- Accepting looser analogies may open up the possibility of detecting more analogies that are inappropriate. For example, Owen's matcher would consider the following as analogous:

  Source: $s1 \cap s2 = s2 \cap s1$
  Target: $s \cap \{\,\} = \{\,\}$

  although their proofs are significantly different. The impact of such loose analogies could be worse on the other approaches than it is on an approach like ours. An approach like ours would soon stop after producing some inappropriate lines. A system that defaults to resolution could however perform a significant amount of searching before stopping (if at all).

- As the descriptions of other systems suggests (see [Kli71, Hal89, Owe90]), the flexible matchers are much more complex than a strict matcher. Indeed, the behaviour of the flexible matchers is defined primarily by their algorithms whereas the behaviour of our matcher is defined by an implicit specification together with a simple exception.

- The flexible matchers are also more expensive computationally. For example, Owen's matcher needs to maintain alternative analogies and uses a best-first searching framework together with a threshold to throw away less promising analogies. Owen sets this threshold level at 0.125 but does not discuss whether the threshold constant is problem dependent, or provide any guidance for setting its level.

In contrast, as we described earlier, our approach is much more cautious. We adopt a relatively strict matcher and then loosen it where appropriate. The primary benefit of using a strict approach is that it gives us some confidence in using an analogous instantiation and rule. In particular if the analogy is strict enough, an analogous instantiation can be applied to the analogous rule so that its conclusion is the target line. However, it remains a challenge for us to loosen our matcher to capture a broader range of analogies in a manner that retains its simplicity and does not open up the possibility of using inappropriate analogies.

To conclude, we hope that this work is a contribution towards the goal of reducing the cost of carrying out formal proofs. In particular, we think that the proof follower reduces the problems of explicitly identifying, abstracting, and coding heuristics.

Acknowledgements

My sincere thanks to Tim Clement for his help and encouragement of this research. Thanks are also due to Richard Moore and Bob Fields for help with *mural* as it was being developed. I would also like to thank both anonymous referees whose comments have improved the presentation of this paper.

References

[B+83] B. G. Buchanan et al. Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors, *Building Expert Systems*, pages 127–167. Addison Wesley, 1983.

[Bun79] Alan Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1979.

[Bun88] A. Bundy. The use of explict plans to guide inductive proofs. In *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988.

[GH80] M. Gick and K. J. Holyoak. Analogical problem solving. *Cognitive Psychology*, 12, 1980.

[Hal89] R. P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39:39–120, 1989.

[JJLM91] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore, editors. mural*: A Formal Development Support System.* Springer Verlag, London, 1991.

[Jon90] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall International, second edition, 1990.

[Kli71] R. E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2:147–178, 1971.

[McD79] John McDermott. Learning to use analogies. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 568–576, Tokyo, 1979.

[Mun81] J. C. Munyer. *Analogy as a means of discovery in problem-solving and learning.* PhD thesis, University of California, Santa Cruz, 1981.

[MW85] Zohar Manna and Richard Waldinger. *The Logical Basis for Computer Programming I.* Addison Wesley, Reading, Massachusetts, 1985.

[Owe90] S. Owen. *Analogy for Automated Reasoning.* Academic Press, 1990.

[Vad92] Sunil Vadera. *Heuristics for Proofs.* PhD thesis, University of Manchester, Manchester M13 9PL, UK, 1992.

[Vad95] S. Vadera. Proof by analogy in mural - a more detailed account. *Formal Aspects of Computing*, 7(E):1–29, 1995.