Change of Representation in Theorem Proving by Analogy

Erica Melis*
Universität Saarbrücken
Fachbereich Informatik 66041 Saarbrücken
email: melis@cs.uni-sb.de

Abstract

Constructing an analogy between a known and already proven theorem (the base case) and another yet to be proven theorem (the target case) often amounts to finding the appropriate representation at which the base and the target are similar. This is a well-known fact in mathematics, and it was corroborated by our empirical study of a mathematical textbook, which showed that a reformulation of the representation of a theorem and its proof is indeed more often than not a necessary prerequisite for an analogical inference. Thus machine supported reformulation becomes an important component of automated analogy-driven theorem proving too.

The reformulation component proposed in this paper is embedded into a proof plan methodology based on methods and meta-methods, where the latter are used to change and appropriately adapt the methods. A theorem and its proof are both represented as a method and then reformulated by the set of metamethods presented in this paper.

Our approach supports analogy-driven theorem proving at various levels of abstraction and in principle makes it independent of the given and often accidental representation of the given theorems. Different methods can represent fully instantiated proofs, subproofs, or general proof methods, and hence our approach also supports these three kinds of analogy respectively. By attaching appropriate justifications to meta-methods the analogical inference can often be justified in the sense of Russell.

This paper presents a model of analogy-driven proof plan construction and focuses on empirically extracted meta-methods. It classifies and formally describes these meta-methods and shows how to use them for an appropriate reformulation in automated analogy-driven theorem proving.

---

*This work was supported by a research scholarship of the Deutsche Forschungsgemeinschaft (DFG)

Contents

1 Introduction **2**

2 Notions and Notation **3**

3 A Model for Analogy-Driven Proof Plan Construction **9**
3.1 Theorem Proving by Analogy 9
3.2 The Model 10

4 Meta-methods **19**
4.1 Meta-methods in REFORMULATION 20
4.1.1 Meta-methods for Normalization 20
4.1.2 Meta-methods for Direct Reformulation 22
4.1.3 Meta-methods for Abstraction 28
4.1.4 Meta-methods for Restructuring 33
4.2 Other Meta-methods 35
4.3 Control Strategies 37

5 Examples **39**
5.1 The Extended Example of de la Tour and Kreitz 39
5.2 Examples from HUA 45

6 Conclusion **52**

1 Introduction

Automated theorem proving systems have attained a remarkable strength when it comes to pure deductive search. They are, however, still weak with respect to long range planning or any other such global search and control issues. Therefore methods and techniques have been suggested that more closely follow the reasoning patterns observed in humans, e.g., by Allen Newell [26] and, more recently, by Alan Bundy [5], who proposed a framework for proof planning to improve the automation of theorem proving.

As evidence for a mathematicians' strong reliance on proof planning we quote Gert Faltings, who discovered the seminal proof for Mordell's Conjecture:

"Man hat Erfahrungen, daß bestimmte Schlüsse unter bestimmten Voraussetzungen funktionieren. Als erstes überlegt man sich daher, wie der Weg aussehen könnte. Man überlegt sich also im Groben: Wenn ich das habe, könnte ich das zeigen und dann das nächste. Hinterher muß man die Details einfügen und sieht, ob man es auch wirklich so machen kann. Es kommt aber auch durchaus vor, daß man mal da sitzt, nicht mehr weiter weiß und dann probiert, wohin der Weg führt" [14][1].

Another important problem solving strategy that can actually be combined with proof planning is the construction of proofs by analogy (see, e.g., Polya [28]), where the known proof of a theorem is often used to guide the proof of an unknown analogous theorem. Gert Faltings reported on his use of analogy during his discovery of the proof for Mordell's Conjecture [14] *"Ich muß sagen, daß ein wesentlicher Teil des Beweises im Prinzip schon da war, den ich nur entsprechend übertragen habe"*[2].

The human use of analogy in reasoning is commonplace, but ill-understood. In particular, the rôle and the use of reformulating the base and the target problem for the analogy formation have found little attention although Indurkhya [20] and Russell [29] established the importance of of reformulation in analogical reasoning in general.

Within mathematics, however, it is well known that constructing an analogy often involves finding the right representation of the theorem, e.g., the right concepts or the right level of abstraction, such that the base and the target problem become similar. Thus, not surprisingly, our empirical study of a mathematical textbook [23] provided ample evidence that an appropriate reformulation of the theorem and its proof is often an important component of human theorem proving by analogy. Consequently, there is a need for a similar but machine supported technique in analogy-driven automated theorem proving that incorporates (heuristically justified) reformulations

[1]translated: We know from experience that certain inferences are usually successful under certain prerequisites. So first we ponder about a reasonable way to proceed to prove the theorem. In other words, we roughly plan: If I would have got a certain result the next result would follow and then the next etc. Afterwards we have to fill in the details, and to check whether the plan really works. But of course, it may happen that you do not see anything and then there is no other way than trial and error.

[2]translated: I should say that basically an important part of the proof was there already, and I only transferred this part appropriately.

of problems and proofs that go beyond the usual symbol mapping techniques of established approaches to analogy (see, e.g., [21, 25, 4, 27].

In this paper we develop a methodology that, in contrast to former approaches to theorem proving by analogy, considers two problems to be *analogous* if there exists a chain of justified reformulations of one problem into the other. Thus in principle we become independent of the actual and often accidental representation of the given theorems. In particular we shall show how the proof plan framework of Alan Bundy (see, e.g., [6]) and the use of reformulations support analogy-driven proof construction. This paper focuses mainly on the formal description of meta-methods that reformulate problems and proofs that are represented as methods.

In the following chapter we develop a methodological framework for our approach to analogy-driven theorem proving. We then summarize briefly the state of the art of analogy in theorem proving and present a model for analogy-driven proof plan construction. The next section presents a set of meta-methods that were actually used for the necessary reformulations in the analogy-driven construction of proofs in an empirical case study [23]. Some examples from a mathematical textbook "Halbgruppen und Automaten" [13] (abbreviated as HUA) and an example of Terry Boy de la Tour and Christoph Kreitz are given in section five. More proofs by analogy are investigated and presented in detail in the case study [23].

2 Notions and Notation

We assume the reader to be familiar with plan-based problem-solving (see e.g., [8]) as well as the main notions of automated theorem proving (see e.g., [22]).

Analogy-driven theorem proving as outlined below presupposes a proof planning process that constructs a proof plan out of a reformulated originally given proof. Hence, our framework is essentially based on the concepts of proof planning from [19] and [6] which are, however, modified. The basic concepts in this framework are *problems, methods*, and *metamethods*.

Problems consist of a theorem and the assumptions the theorem follows from. In the following we shall refer to a proof of a problem, which is to say that this is the proof of a theorem thm from the assumptions ass. Of course we could just write $(ass \rightarrow thm)$ and treat this as a theorem to be shown, but it is often advantageous to handle the assumptions and the theorem separately such as in Natural Deduction. Moreover not all assumptions are initially given in every-day mathematical theorem proving; quite to the contrary most axioms and definitions have to be added as we go along in the proof. In fact this is often the most difficult part of proving, namely to state and find the right assumptions. In our case this will happen in particular for the target problem, when not all relevant assumptions are initially given.

The objective of the proof planning methodology is to find proofs for problems by planning. The result of this planning process is a tree of discrete proof chunks. *Methods* encode these chunks of proofs. They are similar to tactics in proof checking systems as Nuprl [10]. Essentially they are situated between two extremes: As one extreme they may contain fully instantiated and complete proof schemas. The other extreme is that they encode an empty proof schema, i.e., they just consist of a plan line with the theorem to be proved. In between there are those more interesting methods that meet the intuition of a proof idea rather than a full proof.

Meta-methods are then applied to methods in order to reformulate them within the analogy-driven proof plan construction.

More precisely let us take a sorted higher order language which is extended by metavariables for formulae and terms as our object language[3]. In this paper Natural Deduction (ND) [16] is used as the proof calculus. Rules of this calculus are, for instance $(\rightarrow I), (\forall I)$, $(\wedge I)$, and $(\rightarrow D)$ for the introduction of $\rightarrow, \forall$, $\wedge$, and for the deletion of $\rightarrow$ respectively.
Δ, Σ are used for a set of object language formulae. $[x_1, \ldots, x_n]$ denotes the *list* of elements $x_1, \ldots, x_n$. KB is the knowledge base containing axioms, definitions, and already proven theorems. We write $\phi_1 = \phi_2$ for formulae ϕ_1, ϕ_2 if they are equal up to a renaming of their free variables. *Sort declarations* are written as (symbol: sort), for instance, 3: nat or P: problem. $\phi[a/b]$ denotes the formula which results from replacing a by b in the formula ϕ. The methods that originate from a method M_1 or M_2 by some reformulation are denoted as M_{1*}-methods or as M_{2*}-methods respectively.

We say that the postconditions of a method M_1 *match* the postconditions of a method M_2, if $\mathrm{ass}(\mathrm{M}_1) \subseteq \mathrm{ass}(\mathrm{M}_2) \cup$ KB and $\mathrm{concl}(\mathrm{M}_1)=\mathrm{concl}(\mathrm{M}_2)$.

We now define the following notions:

- A **problem** P is a pair that consists of a set of object language formulae, the assumptions ass, and of one object language formula, the conclusion thm. A problem is written as $(ass; thm)$.

- **Methods** play the rôle of plan operators , and they operate on proof trees the nodes of which are sets of problems. They are represented as frame-like data structures where the rows are slots, with slot name and slot filler that look like this:

[3]Throughout this document sans serif font is used for problems and *mathematical* font is used for object level terms and formulae; `typewriter` font indicates metavariables for object level symbols, formulae, and terms and for names of meta-methods. CAPITALIZED font is used for methods except for those methods that have numerical names. PLAN, with and without an index, denotes variables for methods and roman text indicates metalanguage stuff.

Method: `Name of the method`	
parameter	parameters which can be instantiated
pre	preconditions that have to be true for the method to be applicable
post	postconditions that are fulfilled after the method application, e.g., a derived problem
dec-cont	a declarative proof schema
procedure	a schema-interpreting procedure that can be applied to dec-cont
history	contains a trace of certain changes of the method done before for the purpose of their revision

- The parameter slot contains all free variables[4] that occur elsewhere in this method, their instantiation yields a instantiated method.
- Preconditions are sets of problems and postconditions are problems (with the parameters as the only free variables).

  For a method M, **ass(M)** (the assumptions of M) and **concl(M)** (the conclusion of M) denote the first and the second entry of post(M). M is called a *method for a problem* (and vice versa, P a problem of M), if post(M)=P.
- dec-cont contains a proof schema that consists of proof lines which are represented as

  (label$_i$. $ass_i \vdash concl_i$ (METHOD$_i$ $l_{i1} \ldots l_{im}$)).

  label$_i$ is the name of the line. $(ass_i; concl_i)$ is a problem called the core problem, core$_i$, where ass_i is a set of object level formulae and $concl_i$ is an object level formula with the parameters of M as the only free variables. $l_{i1}, \ldots, l_{im}$ are the labels of lines, called *support lines*, to the cores of which METHOD$_i$ is applied. This application should yield the problem $(ass_i; concl_i)$.

  dec-cont(M) can also contain so-called LEMMA-lines the cores of which are elements of pre(M), and METHOD$_i$ = LEMMA.
- procedure provides a program by interpreting dec-cont (see [19] for further details). (For correct methods, this program yields as output the postcondition when the preconditions were input.) In [5] methods contain this program itself. Here and in [19] the declarative description and its interpreter are separated, such that one can reason about and reformulate the declarative content without having to change the interpreting procedure.
- History contains a trace of changes by certain meta-methods.

Since the dec-cont of a method may contain names of methods, they are to be defined recursively.

[4] for functions, relations, formulae, and terms

- The *basic methods* M correspond to the rules of the proof calculus (here: Natural Deduction rules). Their declarative content contains just one line which is not a LEMMA-line, and the method name is the name of a calculus rule such as for example,
  $$\frac{(\Delta_1;H_1),(\Delta_2;H_2)}{(\Delta_1\cup\Delta_2;H_1\wedge H_2)}.$$
  For later reference, the general form of such a rule is
  $$\frac{(\Delta_1;H_1),\ldots,(\Delta_n;H_n)}{(\Delta;H)}.$$
  This rule is applied to the preconditions of M.
- For *Methods* M are basic methods or methods such that
  1. several method names M_i occur in the lines of dec-cont(M) that were defined before, or
  2. dec-cont(M) contains variables for methods (denoted by PLAN_i). This indicates that the method which should yield core_i is still unknown.

  The occurrence of a method name M_i in a line $(l_i.\ ass_i \vdash concl_i\ \ (\mathrm{M}_i\ l_{i1}\ldots l_{im}))$ of dec-cont(M) means that M_i is applied to the cores of the lines $l_{i1}\ldots l_{im}$ (and the result is $ass_i \vdash concl_i$ for verified methods).

A basic method M is called *verified*, if the application of the corresponding calculus rule to its preconditions correctly yields its postcondition. That is, if there exists an instance $\sigma(R)$ of the calculus rule R such that
pre(M)=$\{\sigma(\Delta_1;H_1),\ldots,\sigma(\Delta_n;H_n)\}$ and
post(M)=$\sigma(\Delta;H)$.

A non-basic method M is called *verified* if

- no method variable occurs in M,
- all methods, the name of which occur in dec-cont(M), are verified,
- for every non-LEMMA-line $(l_i.\ ass_i \vdash concl_i\ \ (M_i\ l_{i1}\ldots l_{im}))$ there exists a substitution σ of parameters of M_i such that $\sigma(\mathrm{ass}(\mathrm{M}_i)) \subseteq ass_i$,
  $\sigma(\mathrm{concl}(\mathrm{M}_i)) = concl_i$, and
  $\sigma(\mathrm{pre}(\mathrm{M}_i)) = \{\mathrm{core}_{i1},\ldots,\ \mathrm{core}_{im}\}$ for lines $l_{i1}\ldots l_{im}$ preceding l_i in dec-cont(M), and
- the method's application is guaranteed to yield its postcondition when its preconditions are fulfilled, i.e., post(M)=$core_l$ for the final line l_l.

Methods may somehow represent *proof ideas* as they can have PLAN-lines and need not be fully instantiated. Let us, for example, look at an instance of the method `hom1` [19]:

Method: `hom1`	
parameter	`formula`$_f$, `f`: function
pre	
post	$\{ass(1), ass(2), ass(3)\}; symmetric(\mathtt{f}(\rho))$
dec-cont	1. ; 1 $\vdash \forall x$ `formula`$_f$ (HYP) 2. ; 2 $\vdash \forall\sigma(symmetric(\sigma) \leftrightarrow \forall x, y((x,y) \in \sigma \rightarrow (y,x) \in \sigma))$ (HYP) 3. ; 3 $\vdash symmetric(\rho)$ (PLAN$_1$) 4. ; 1, 3 $\vdash \forall x, y((x,y) \in \mathtt{f}(\rho) \rightarrow (y,x) \in \mathtt{f}(\rho))$ (PLAN$_2$) 5. ; 1, 2, 3 $\vdash symmetric(\mathtt{f}(\rho))$ (METHOD$_5$ 2 4)
procedure	schema-interpreter
history	

The proof idea of this method is to prove $symmetric(\mathtt{f}(c))$ from certain assumptions and from the definition of `f` in line 1 in the following manner:

- First prove that $symmetric(\rho)$ by some method.
- Then show that $\forall x, y((x,y) \in \mathtt{f}(\rho) \rightarrow (y,x) \in \mathtt{f}(\rho))$ is provable from the definition of `f`, $symmetric(\rho)$, and possibly other assumptions.
- Finally prove the conclusion $symmetric(\mathtt{f}(\rho))$ from lines 4 and 2 by the application of METHOD$_5$.

Of course, this method is not as complicated as Falting's proof idea for Mordell's Conjecture, but it is quite similar in kind.

Methods can also explicitly represent the structure of a proof by combining several simple methods to more complex methods.

Methods have not to be fully instantiated (although most of our base methods are). and therefore they can be used to represent methods in the usual mathematical sense, as for example, Cantor's diagonalization method. The very same method can represent several analogical proofs or proof parts.

- **Proof plans** correspond to the usual plans in plan-based problem solving, and they are built from finitely many methods and a substitution σ of the methods' parameters. Let M_i and M_j be methods that occur in a proof plan Mj *succeeds* M_i, if $\sigma(\text{post}(M_j)) \subseteq \sigma(\text{pre}(M_i))$ (see figure 1).

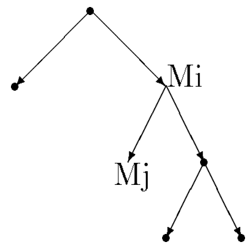


Figure 1: Method M_j succeeds method M_i

For finally proving a problem, the goal is to construct a plan of a problem as complete as possible.

A *complete plan of a problem* P

- contains verified methods only,
- has only one root method M and we have $\sigma(\text{post}(\text{M}))=\mathsf{P}$. For the leaf methods M_{1i} we have $\text{pre}(\text{M}_{1i})=\emptyset$, and
- for each method $\text{M}_m$of the proof plan which is not a leaf there exist succeeding methods $\text{M}_{m1}, \ldots, \text{M}_{mk}$ in the proof plan such that $\sigma(\text{pre}(\text{M}_m)) = \bigcup_{i=1}^{k}(\sigma(\text{post}(\text{M}_{mi}))$.
  Hence a complete proof plan is a tree.

Methods and proof plans are used for different purposes: Methods represent the proof parts to be reformulated. Proof plans are used to combine methods via pre- and postconditions finally to a complete proof.

- **Meta-methods** map sets of methods to other sets of methods, for example by splitting one method into several new methods, or by combining methods to a new method. If a meta-method maps a set of just one method to another singleton we simply say that the meta-method maps a method to a method.

  Meta-methods are frame-like data structures containing the following slots:

Metamethod: `Name`	
parameter	to be specified for a particular application
pre	preconditions for the application
post	result of the application
procedure	procedure for the application
rating	rating procedure

– Problems, terms, symbols, and history slots of methods may occur as parameters of a meta-method.

– If the preconditions are fulfilled the meta-method is applicable. The preconditions are formulated in a metalanguage that describes methods.

– The postconditions describe the resulting method in the same metalanguage as the preconditions.

– Rating is a procedure that computes an actual rating. Reformulations should be justified at least heuristically. The rating procedure takes the justification and situation information as an input and computes a rating for the application of the meta-method in the current situation. Rating as a procedure located in the meta-methods, has been favoured above pure justification, as an information to be used by the planner, since the set of meta-methods can be extended without changing the planner. There are several kinds of justification e.g., the mathematical theory in which the meta-method should work, dualities of theories, semantics of the used concepts such as homomorphism etc. The actual ratings are not properly developed currently.

3 A Model for Analogy-Driven Proof Plan Construction

3.1 Theorem Proving by Analogy

Problem solving by analogy (see e.g. [7, 18, 11]) means to find a solution for a target problem on the basis of a given solution of a given base problem which is somehow similar – or analogous – to the target problem.

The specific subject of theorem proving by analogy is a more formal instance of this general pattern of reasoning. As before let a problem $(ass; thm)$ be a pair of assumptions and a theorem and a proof of a problem is a proof of $ass \vdash thm$. Then the task is to find a proof for a given target problem P2 based on a given proof of a given base problem P1 which is supposed to be similar to P2.

Robert Kling's work [21] on theorem proving by analogy was one of the first attempts in the field. His system ZORBA essentially produced mappings between predicate symbols (and variables). The mapping was applied to the assumptions of the base proof in order to find analogous assumptions for the target case. Thus the set of formulae that could serve as assumptions in the analogous proof was very limited in this approach.

J.C. Munyer's [25] focus is on the formulae derived in each proof step. He applied a symbol mapping to these formulae in order to obtain the corresponding formulae in the desired analogous proof. The symbol mapping was obtained from the given theorems.

A very elaborate work on analogy in theorem proving is that of Stephen Owen [27],who thoroughly analyzed and advanced the approaches of R.E. Kling and J.C. Munyer and showed that Kling's and Munyer's accounts were inadequate even for simple analogies.

His emphasis is placed on a matcher that recursively constructs symbol mappings and argument pairings. This mapping makes the terms of the base and the target theorems equal. "As matching proceeds, symbolic associations will be made between the head symbols of subterms which are associated. This association is added to the existing mappings." His system is based on two inference rules, binary resolution and paramodulation. To construct an analogue to the given proof, his system constructs the formulae in the target proof from the inferred formulae in each calculus step of the base proof. It also generates the assumptions for the target proof steps from the assumptions of the base proof steps by extending the above symbol mapping. This extension is achieved by six rules of different strength based on heuristics and justifications for these rules. His system can cope to some extend with mappings which do not generate a correct target proof, e.g., by a special means end analysis that guide the search for missing proof steps.

Another computational simulation of theorem proving by analogy has been carried out by Woody Bledsoe and his students Bishop Brock, Shaun Cooper, and William Pierce [4]. Their examples were mainly taken from the theory of real analysis. An initial symbol mapping and a pairing of definitions, lemmata and already proven theorems have to be provided by the user. Their work then focuses on failed constructions of analogous proofs where heuristics fill the gaps in the transformed proof and recognize subproofs. For example, B. Brock et al. [4] suggest a "Double Entry Fetching" operator that attempts the application of the next inference rule of the base proof by finding a bridge lemma which links the current clause and the rule to be applied.

In summary, computational accounts of theorem proving by analogy have been more or less dominated by the idea to map symbols of the base problem (and its proof) to the symbols of the target and then to construct the desired proof from the given proof at the calculus level (e.g., resolution) by a translation of each single proof step.

3.2 The Model

The following model encapsulates our experience of proving all theorems in the textbook HUA that used analogy. This book on Automata Theory is well-known and used as a standard text for undergraduates in many computer science departments in Germany. It contains several proofs by analogy based on reformulation rather than just symbol mapping. The model enables to cope with analogies based on the transfer of proof methods and proof ideas as well.

The model for our analogy-driven proof plan construction consist of the following procedures: Initialize, Reformulation, Match, Reversion, Verification, and Proof

Planning, to be explained below.

Starting with the method M_1 made up from the base problem P1 and its proof, and method M_2 made up from the target problem P2 without a proof (i.e., dec-cont(M_2) is just a single PLAN-line), the goal is to reformulate M_1 to a method M_{1k} in k steps, such that the postcondition of M_{1k} matches P2. Figure 2 outlines the process of reformulation of a method M_1 into a method M_{1k} whose postcondition matches P2 of method M_2.

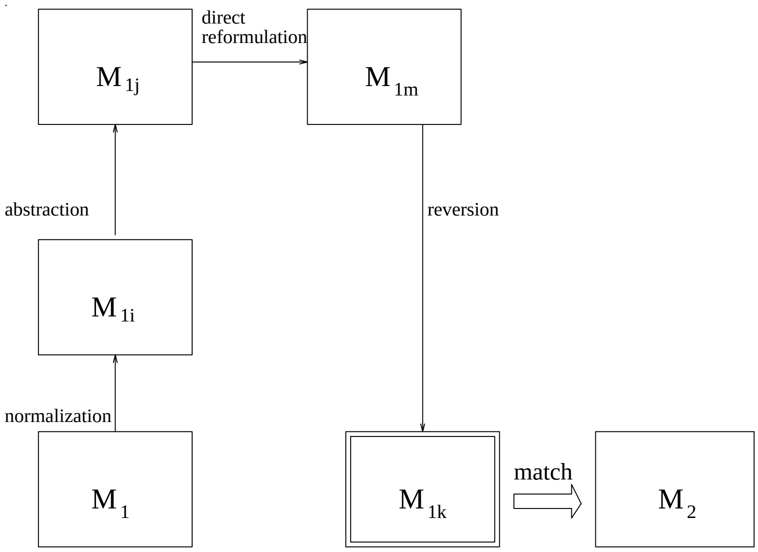


Figure 2: Outline of the reformulation

In addition to the process as sketched in figure 2 there is a preparation procedure for the verification of methods, which removes the method variables from M_{1k} and yields the method M_{1r}. This method M_{1r} is then checked by a verifier and if the verification succeeds, M_2 is replaced by M_{1r}, which finally contains a verified proof schema.

If the verification fails, the same process is tried again but with all the sub-methods, sub-submethods etc. of M_1 and M_2. These sub-methods are obtained from the structuring reformulations presented below.

When all methods and sub-methods have been dealt with, there is a set $\{M_{1r}^i\}_i$ of methods that were obtained from M_1 by some reformulations, such that their post-conditions match P2 or some subproblems of P2. From these M_{1r}^i we try to build a proof plan for P2. The plan should be as complete as possible, hence additional methods are often necessary to fill the gaps between the preconditions and the post-conditions of the different M_{1r}^i. Figure 3 shows the idea of this analogy-driven proof plan construction: Several sub-methods of M_1 can be reformulated such that their postconditions match a subproblem of P2. These reformulated methods are then the elements of a new proof plan for P2.

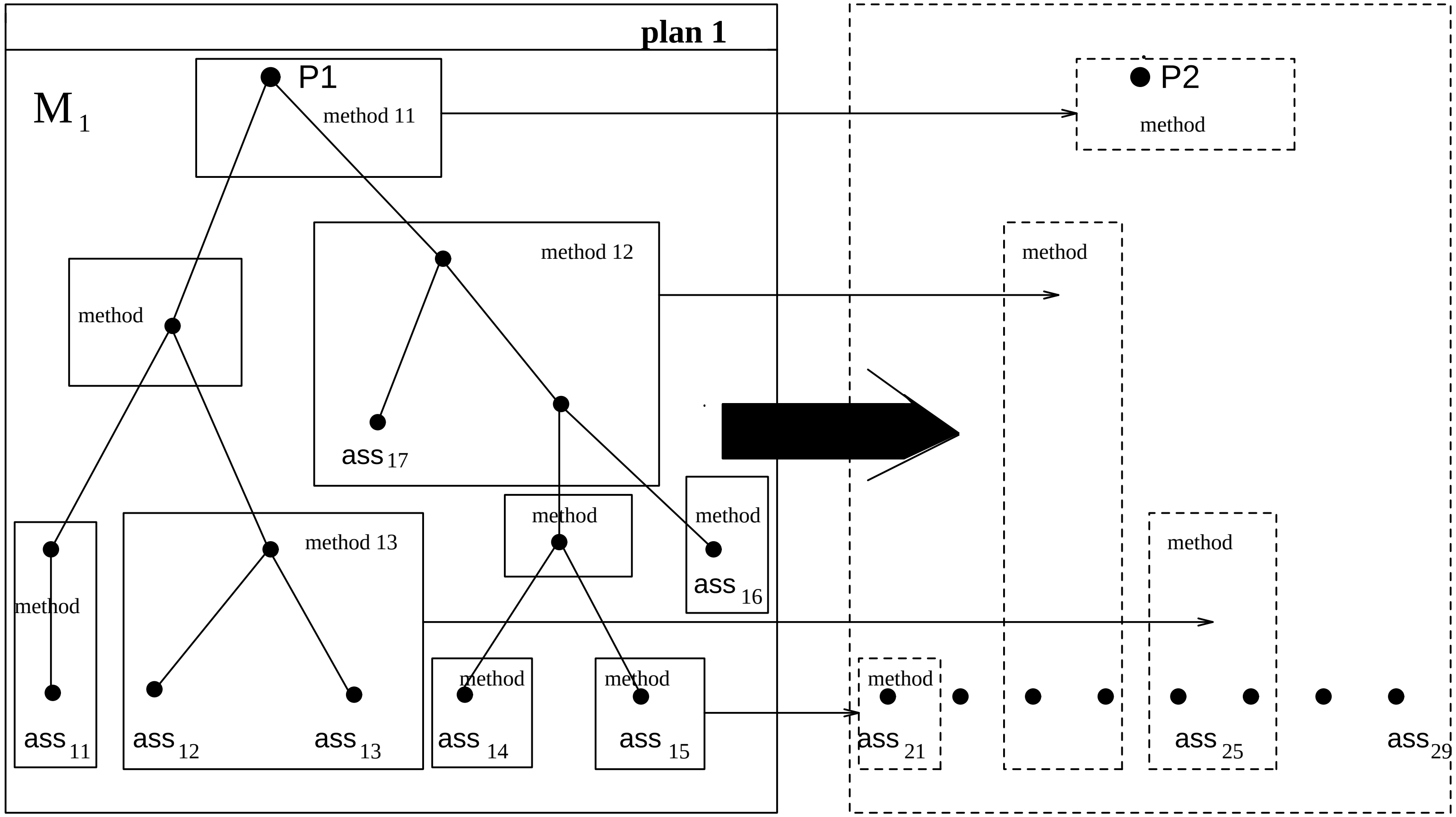


Figure 3: Outline of transforming and replacing methods

The individual steps in the analogy-driven proof plan construction are carried out by the following procedures:

1. **INITIALIZE**
For proving a target problem P2=$(ass2; thm2)$ by analogy to the proof of a given base problem P1=$(ass1; thm1)$, INITIALIZE sets up the method M_1, that contains the known proof in its dec-cont slot, and the *initial* method M_2 for P2. An *initial* method M for a problem P=$(ass; thm)$ is a method whose declarative content contains for all $A \in ass$ a line $(\ldots A \vdash A$ (HYP)), and a PLAN-line $(\ldots ass \vdash thm$ (PLAN)). It represents the general goal "prove thm under the assumption of ass." The following sketches such an initial method in general:

Method: `INITIAL(P)`	
parameter	
pre	
post	$(ass; thm)$
dec-cont	1. ; 1 $\vdash$ A (HYP) 2. ; $\vdash$ ... (HYP) 3. ; 1 $\vdash$ thm (PLAN)
procedure	schema-interpreter
history	

The procedure INITIALIZE also sets up several parameters of the algorithm CONSTRUCT_PROOF_BY_ANALOGY of analogy-driven proof plan construction which is given below: the methods that have to be treated as `current_methods`, the active_reformulation class (`active_ref`), the set of methods already treated by the active reformulation class (`treated_methods`), the set of submethods which result from the last structuring `sub-methods`, and the set of revised matched methods (`matched_methods`).

2. **MATCH**
This procedure tries to match the postcondition of a current M_{1*}-method M_{1i} with the postcondition of a current M_{2*}-method M_{2i} such that
ass(M_{1i}) $\subseteq$ ass(M_{2i})$\cup$ KB and concl(M_{1i})=concl(M_{2i}). As the definition shows, the matching is tolerant for missing axioms that may be stored in a given knowledge base KB.

3. **REFORMULATION**
If no match can yet be obtained for the current M_{1*}-methods they are subjected to REFORMULATION, which is achieved by the application of meta-methods.

Although these reformulations could in principle be limited to P1-methods, such that P1=$(ass1; thm1)$ is reformulated to a problem $(ass1'; thm1')$ with $thm1' = thm2$ and $ass1' \subseteq ass2 \cup$ KB, it is more convenient to apply normalizing and abstracting meta-methods to both M_{1*}- *and* M_{2*}-methods. Such reformulations are advantageous since they are more purpose directed: It is easier to abstract two methods and then to find an additional reformulation that yields a problem that matches the abstracted problem, than to find an abstraction, a reformulation, and a reverse abstraction that provide a problem matching the original P2. In the former case it is easy to find out which reverse abstraction to use. Also the reformulation of M_{1j} to M_{1m} is more goal directed (see figure 4).

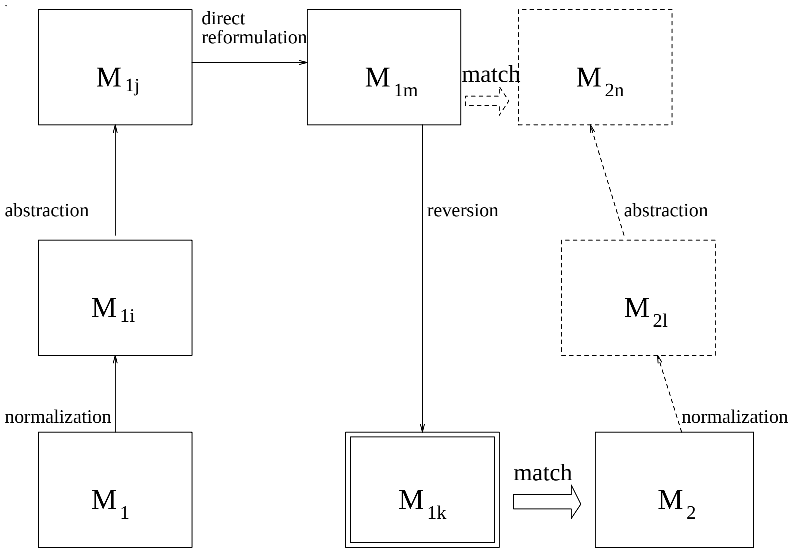


Figure 4: Outline of the more goal directed reformulation

If the postcondition of some M_{2*}-methods are not matched by the postcondition of a verified M_{1*}-method, then structuring meta-methods are applied that split the current M_{1*}- or M_{2*}-methods. The goal of structuring is to produce sub-methods which are treated by the algorithm then in the same way as the original methods.

4. **REVERSION**
Once an M_{1*}-method is successfully matched, it is reformulated to a new M_{1*}-method by the reversion of all entries stored in the history slot of the matched M_{2*}-method during REFORMULATION. Thus abstraction and normalization steps which were applied to the original M_{2*}-method are made undone by REVERSION. The postconditions of the revised M_{1*}-methods should then match the original problem P2 or a subproblem of it.

5. **VERIFICATION**
Redundant lines in the declarative content of the current (revised) M_{1*}-methods are deleted, PLAN-lines are changed and method variables are removed, thus preparing methods for their actual verification. The deletion of method variables produces additional preconditions in some methods and later, during proof planning, these preconditions cause a change of the proof plan by the insertion of additional methods.

   It is then checked whether the method M at hand is verified. If it is, then all those M_{2*}-methods M' are replaced by M and removed from `current_methods`, whose postconditions match post(M). All M_{2*}-methods that descended from M' are removed from `current_methods` as well. If M is not verified, then the corresponding M' is further processed but M is a candidate to be used in PROOF PLANNING nevertheless.

6. **PROOF PLANNING**
This process is not treated in this paper. It is a usual proof planning procedure [5]. It operates mainly on the set of methods which have passed VERIFICATION and takes them as preferred candidate elements for the proof plan. Verified methods are favoured candidates, compared to methods that have the same postcondition but are not verified. (Alternatively we think about a stepwise Proof Planning that is carried out after each succesful match.) Proof Planning tries to partially order these methods by comparing instances of their pre- and postconditions respectively. It can use information from the structuring of the M_{1*}- and M_{2*}-methods. Proof Planning starts with a method M that has the desired problem `P2` as its postcondition. Then it looks for methods that have problems of pre(M) (maybe less instantiated) as its postcondition etc. The process stops when the preconditions of the new methods are empty or there are no new methods. It may provide several proof plans.

   Often there will still be gaps between the elements of the proof plan. That is, not all preconditions of a method are found in the succeeding methods. Hence, to obtain a plan as complete as possible, additional methods have to be inserted which can be found, for instance, by searching bridge lemmas or by difference matching, see [9, 1].

The following gives the basic algorithm

```
CONSTRUCT_PROOF_BY_ANALOGY

Input: problems P1, P2, and proof of P1
Output: proof plan for P2

1.   INITIALIZE
2.   MATCH
     if (M1i, M2i do not match)
          then go to 5a
    update matched_methods
3.   REVERSION
4.   VERIFICATION
     update current_methods
5.a  if (treated_methods = current_methods)
          then go to 5c
  b  choose M1i and M2i of current_methods not treated by active_ref
     add these methods to treated_methods
     if (active_ref of the chosen methods possible)
          then go to 5d
          else  go to 5a
  c  if (active_ref = structuring)
          then
               if (sub-methods = empty)
                         then  go to 6
                         else  sub-methods:= empty
          else active_ref := NEXT(active_ref)
               treated_methods:= empty
               go to 5b
  d update current_methods
     REFORMULATION
     if (active_ref = structuring)
          then add the new methods to sub-methods
    update current_methods
     go to 2
6.   PROOF PLANNING
     Stop
```

Figure 5: algorithm of analogy-driven proof plan construction

An Example

As an illustration of the previous algorithm consider the following example, which is presented in more technical detail later on in section 5. This example was used by de la Tour and Kreitz [12], who tried to prove $Theorem2$ by analogy to the proof of $Theorem1$, where:
$Theorem1$: $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb) \rightarrow \exists x Pbx)$ and
$Theorem2$: $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \rightarrow \exists x Pcx)$.

All transformations of the proof of $Theorem1$ to the proof of $Theorem2$ are explicitly given in [12] by the instruction of the following kind "rename constant b to constant c and repeat a certain given proof step twice." The proof of $Theorem1$ and this instruction were both represented as typed formulae using the Curry-Howard-Isomorphism. Let us look at this example within our framework of analogy-driven proof plan construction. The transformation steps are as follows.

1. INITIALIZE
   The M_{1*}-method M_1, which is built from $Theorem1$ and its proof, is verified and has the known ND-proof as dec-contM_1). Its postcondition is
   $\mathsf{P1}$=($\emptyset$;$Theorem1$), i.e., ass(M_1)=$\emptyset$ and concl(M_1)=$Theorem1$.
   The preconditions are empty. The history slot is empty.

   The first M_{2*}-method M_2=INITIAL(P2) for $\mathsf{P2}$=($\emptyset$;$Theorem2$) is built up with post(M_2)=$\mathsf{P2}$ and a declarative content consisting of the line
   ($l.$ $\emptyset \vdash Theorem2$ (PLAN)).

2. MATCH
   First time there is no match of the current methods. Hence, no reversion and verification is possible.

3. REFORMULATION
   The reformulation steps are:

   - The normalization process, corresponding to the application of the deduction theorem, reformulates M_1 to M_{11} with concl(M_{11})=$\exists x Pbx$ and ass(M_{11})=$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, (Paa \vee Pbb)\}$.
     dec-cont(M_{11}) differs from dec-cont(M_1) in that the last line
     $(\ldots \emptyset \vdash \forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb) \rightarrow \exists x Pbx)(\rightarrow I))$
     is missing.
     The history slot now gets an entry for this normalization.

     A similar normalization reformulates M_2 to M_{21} with
     ass(M_{21})=$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, Qbc, (Paa \vee Pcc)\}$ and concl(M_{21})=$\exists x Pcx$.
     The only line of dec-cont(M_2)
     $(\ldots \emptyset \vdash \forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \rightarrow \exists x Pcx \ldots)$
     is changed to a line

$(\ldots\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, Qbc, (Paa \vee Pcc)\} \vdash \exists x Pcx$ (PLAN)).
The history slot gets an entry for this normalization.

The postconditions of the methods M_1 or M_{11} still do not match the postconditions of M_2 or M_{21}.

- So first it is tried to match the conclusions. The symbol mapping $\{b \mapsto c\}$ applied to M_{11} yields M_{12} with ass(M_{12})=$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qac, (Paa \vee Pcc)\}$ and concl(M_{12})=$\exists x Pcx$. Then concl(M_{12}) = concl(M_{21}).
- Now there are still differences concerning the assumptions of M_{12} and M_{21}: They differ in the subsets $\{Qac\}$ and $\{Qab, Qbc\}$. Now the preconditions of one of our meta-methods are met. It reformulates M_{12} to M_{13} with ass(M_{13})=$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, Qbc, (Paa \vee Pcc)\}$ and concl(M_{13})=$\exists x Pcx$.
  This meta-method replaces the line
  $(\ldots \Delta \cup \{Qac\} \vdash \forall z(Paz \rightarrow Pcz)\ (\forall D, \rightarrow D, \ldots))$
  of dec-cont(M_{12}) by the line
  $(\ldots \Delta \cup \{Qab, Qbc\} \vdash \forall z(Paz \rightarrow Pcz)$ (PLAN)) in dec-cont(M_{13}).

4. MATCH
   The postcondition of M_{13} now matches the postcondition of M_{21}.

5. REVERSION
   Reverse-normalization with respect to the history slot of M_{21} applied to M_{13} yields M_{14} with ass(M_{14})=$\emptyset$ and
   concl(M_{14})=$\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \rightarrow \exists x Pcx$.
   dec-cont(M_{14}) has the last line:
   $(\ldots \emptyset \vdash \forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \rightarrow \exists x Pcx\ \ (\rightarrow I \ldots)$.

6. VERIFICATION
   To obtain a verified method, the method variable (PLAN) is to be removed: A meta-method applied to M_{14} yields M_{15}. It provides additional preconditions. M_{15} can be verified.

   M_2 is replaced by M_{15}. Figure 6 shows this process of reformulation.

7. PROOF PLANNING
   In completing the proof plan, an additional method has to be inserted which has pre(M_{15}) as postcondition.

   Knowing the postcondition $(\Delta \cup \{Qab, Qbc\} \vdash \forall z(Paz \rightarrow Pcz))$, the proof becomes (slightly abbreviated)
   $\Delta, Qab \vdash \forall z(Paz \rightarrow Pbz)$
   $\Delta, Qbc \vdash \forall z(Pbz \rightarrow Pcz)$

$\Delta, Qab, Qbc \vdash \forall z(Paz \rightarrow Pcz)$
is a candidate for the declarative content of the additional method.

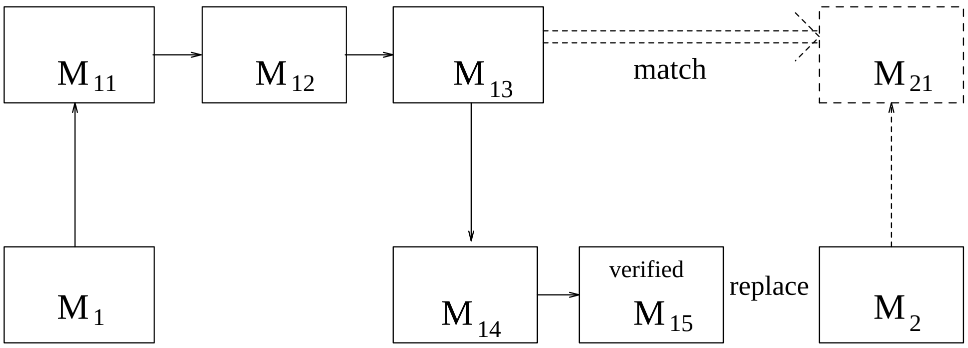


Figure 6: The reformulation of M_1 and M_2

This plan transformation is pretty similar to human attempts to prove $Theorem2$ by analogy, and it is interesting to note that this example can be handled in exactly the same way as for example the proof of theorem 17.6 in HUA.

4 Meta-methods

The following selection of meta-methods is the result of an empirical study of the mathematical textbook [13], where we isolated all those proofs that are explicitly mentioned as proofs by analogy. These examples are presented in a case study [23], that contains the Natural Deduction proofs. The following set of meta-methods is sufficient for an automated construction of these proofs by analogy, however, it is, of course, not complete in general.

The presented meta-methods turned out to be also useful for the manipulation of methods as part of the proof planning paradigm, e.g., for the generation of new abstracted or generalized methods and for structuring of methods. Some of them correspond to standard techniques in automated theorem proving. Note that some metamethods correspond to methods with a similar purpose, e.g., conjunction splitting. But the fundamental difference between these methods and meta-methods is that methods operate on a partial proof tree (or forest) and meta-methods operate on methods.

Some meta-methods are described only informally rather than representing them fully by the appropriate frame schemata.

4.1 Meta-methods in REFORMULATION

We distinguish several classes of meta-methods ; they are classified with respect to their different usage and effect:

- meta-methods for normalization
- meta-methods for direct reformulation
- meta-methods for abstraction
- meta-methods for restructuring

We shall give some examples of the application of these meta-methods. Some treatments (subproblems of 7.5.7, 5.7, and 17.6 of HUA, and the de la Tour and Kreitz' example) are given in full detail in section 5.

In the following the method a meta-method is applied to is denoted by M and the resulting method is denoted by M'.

4.1.1 Meta-methods for Normalization

The purpose of these meta-methods is to make the postconditions of methods comparable in that they produce normal forms which are often used (but seldom mentioned) in mathematics. Meta-methods from this class are generally applicable to M_{1*}- *and* M_{2*}-methods. They fill the history slot of methods. They do not provide new parameters for the methods and they do not delete any information from the methods.

Most of the meta-methods for normalization are based on well-known facts and they are rather trivial in comparison to some of the later meta-methods.

Among the most important normalizing meta-methods that we have used are the following:

Decompose Assumptions

This meta-method changes the assumptions ass(M) of a method.

- It replaces the formula $(\phi_1 \wedge \ldots \wedge \phi_n)$ by $\phi_1, \ldots, \phi_n$ in ass(M), i.e. it just brakes up the conjunctions, if there are any. Furthermore it also changes the slot filler of dec-cont(M) by the following rules:
- Remove the line $(l.\ \emptyset \vdash \phi_1 \wedge \ldots \wedge \phi_n$ (HYP)) from dec-cont(M).
- Replace the lines $(l_i.\ \{(\phi_1 \wedge \ldots \wedge \phi_n)\} \vdash \phi_i\ \ (\wedge D \ldots))$ of dec-cont(M) by lines $(l_i.\ \{\phi_i\} \vdash \phi_i$ (HYP)).
  Replace $(l_{ij}.\ \{(\phi_1 \wedge \ldots \wedge \phi_n)\} \vdash \phi_i \wedge \phi_j\ \ (\wedge D \ldots))$ by
  $(l_{ij}.\ \{\phi_i, \phi_j\} \vdash \phi_i \wedge \phi_j\ \ (\wedge I l_i, l_j))$. (The same for more than two conjuncts.)
- Replace the assumption $(\phi_1 \wedge \ldots \wedge \phi_n)$ in all lines of dec-cont(M) by the assumptions $\phi_1, \ldots, \phi_n$.

Handle Definitions

Definitions of a predicate P or a function f are formulae of the form
$\forall x_1, \ldots, x_n(P(x_1 \ldots x_n) \leftrightarrow formula_P(x_1 \ldots x_n))$ and
$\forall x_1, \ldots, x_n y(f(x_1 \ldots x_n) = y \leftrightarrow y = term(x_1, \ldots, x_n))$, where $formula_P$ is an object level formulae that defines P and $term_f$ is an object level term that defines f.
For example:
$\forall x_1, x_2(symm(R) \leftrightarrow ([x_1, x_2] \in R \rightarrow [x_2, x_1] \in R))$ is the definition of symmetry for the relation R.
Expanding a definition then amounts to rewriting the occurrence of, say
$P(c_1, \ldots, c_n)$ by $formula_p(c_1, \ldots, c_n)$ and $f(c_1, \ldots, c_n)$ by $term(c_1, \ldots, c_n)$ respectively.

The assumptions of a problem and its analogue may differ in the form of definitions, if one contains an expanded definition for a certain predicate P and the other contains a corresponding formula $P(c_1, \ldots, c_n)$. To remove these superficial differences, `Handle Definitions` enlarges ass(M) by the definitions of the predicate and function symbols occurring in the postconditions of M.

Furthermore the meta-method changes M_{2*}-methods by adding the expanded definition that corresponds to the formula $P(c_1, \ldots, c_n)$ or $f(c_1, \ldots, c_n)$ to ass(M) if this formula belongs to ass(M) and adds the formula if the expanded definitions belong to ass(M). The content of the slots pre(M)and dec-cont(M), and concl(M) are not changed.

The expansion could lead to the well-known explosion of expanding the expansion of a definition (and so on), but note, that we only expand up to the heuristic limit of a recursion depth of two.

Deduction Normal Form

`Deduction Normal Form` reformulates a method in order to obtain conclusions that are comparable. This meta-method corresponds to and is justified by the deduction theorem.

`Deduction Normal Form` is applied to M_{1*}- and M_{2*}-methods M if
a) concl(M) is of the form $A \rightarrow B$ or
b) concl(M) is of the form $\forall x(A(x) \rightarrow B(x))$.
For ass(M)=Δ the resulting method M' asserts A and concludes B, i.e., ass(M')=$\Delta \cup \{A\}$ and concl(M')=B or ass(M')=$\Delta \cup \{A(c)\}$ and concl(M')=$B(c)$.

For instance, in the example of de la Tour and Kreitz, the method M_1 with
concl(M_1)=$\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb) \rightarrow \exists x Pbx$
and ass(M_1)=$\emptyset$ is reformulated by `Deduction Normal Form` to the method M_{11} with concl(M_{11})=$\exists x Pbx$ and
ass(M_{11})=$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, (Paa \vee Pbb)\}$.

Let ass(M)=Δ and let c be a constant. The reformulation of the method M proceeds as follows:

1. Change lines of dec-cont(M)

   (a) If there are lines
   a) (l. $\Delta \vdash A \rightarrow B$ $(\rightarrow Il_s)$) or
   b) (l_t. $\Delta \vdash A(c) \rightarrow B(c)$ $(\rightarrow Il_s)$) and
   (l. $\Delta \vdash \forall x(A(x) \rightarrow B(x))(\forall Il_t)$)
   in dec-cont(M), then delete these lines.

   (b) If there is a line
   a) (l. $\Delta \vdash A \rightarrow B(\mathrm{PLAN}_1)$) or
   b) (l. $\Delta \vdash A(c) \rightarrow B(c)(\mathrm{PLAN}_1)$) in dec-cont(M), then replace it by
   a) (l. $\Delta \cup \{A\} \vdash B(\mathrm{PLAN}_2)$)
   b) (l. $\Delta \cup \{A(c)\} \vdash B(c)(\mathrm{PLAN}_2)$) with a new method variable PLAN_2
   and add the line
   a) (l_r. $\{A\} \vdash A$ (HYP))
   b) l_r $\{A(c)\} \vdash A(c)$ (HYP).

   (c) Else, insert the two lines
   a) (l_{n+1}. $\{A\} \vdash A$ (HYP)) and
   (l_{n+2}. $\Delta \cup \{A\} \vdash B$ $(\rightarrow Dl, l_{n+1})$) or
   b) (l_{n+1}. $\{A(c)\} \vdash A(c)$ (HYP)) and
   (l_{n+2}. $\Delta \cup \{A(c)\} \vdash B(c)$ $(\rightarrow Dl, l_{n+1})$) with a new constant c.

2. Rewrite ass(M) and concl(M) to
   a) ass(M')=$\Delta \cup \{A\}$ and concl(M')=B
   b) ass(M')=$\Delta \cup \{A(c)\}$ and concl(M')=$B(c)$.

3. If the antecedens of concl(M) is a conjunction $A = (A_1 \wedge \ldots \wedge A_m)$, then set ass(M')=$\Delta \cup \{A_1, \ldots, A_m\}$ and in addition change the declarative content as in `Decompose Assumptions`.

4. (Ded: label;$A \rightarrow B$) is stored in the history slot with some of the labels a), b), 1(a), 1(b), 1(c), and Conj which indicate the kind of treatment of dec-cont(M). This information makes the reversion of `Deduction Normal Form` possible later on.

4.1.2 Meta-methods for Direct Reformulation

The meta-methods of this class reformulate M_{1*}-methods only. They have a problem P as a parameter which is the postcondition of the M_{2*}-method the reformulation of the method M is directed to and they do not change the history slot of M.

One problem that is still unsolved is that these reformulations are directed towards a problem P as above. In our context of theorem proving by analogy this problem is the target problem $\mathsf{P2}$ or normalizations and abstractions thereof. That means that at most one direct reformulation can be applied and this may be insufficient in many cases.

Symbol Mapping

The mapping of symbols from the base case to the target has been the standard approach in theorem proving by analogy, where this mapping establishes the analogy between two proofs. The representation of a symbol mapping as a meta-method allows us to integrate this into our framework. The meta-method `Symbol Mapping` tries to make the conclusions of two methods equal[5].

Metamethod: `Symbol Mapping`	
parameter	symbols, P: problem
pre	P=(ass, thm) and there exists a mapping f:{symbols} $\mapsto$ {symbols} such that for symbols $symbol_i$ concl(M)$[symbol_i/\mathrm{f}(symbol_i)]_i] = thm$
post	concl(M')=thm
procedure	PROCSYMBOL (see below)
rating	SYMBOL-rating

The mapping f is selected according to heuristics that are encoded in SYMBOL-rating. We require that the mapping f is in fact a function, i.e., if $\mathrm{f}(x) = y$ and $\mathrm{f}(x) = z$, then $y = z$. Some authors call this consistency and they also sometimes explicitly drop this requirement (see e.g., [27]). Furthermore, f should be minimal in that it maps only symbols occurring in concl(M). If more than one symbol mapping is applicable, then the heuristic measure of the justifications rate the different mappings. Several such heuristics were proposed by Owen in [27], e.g., the identical symbols heuristic and the syntactic type heuristic. A good justification of `Symbol Mapping` is given by the so-called theory interpretations that are employed in IMPS [15].

PROCSYMBOL modifies M:

- Replace all occurrences of the symbol s by $\mathrm{f}(s)$.
- Change, if necessary, the corresponding sort declarations.

Examples

Examples for an application of `Symbol Mapping` are the reformulation of the method for theorem 3.3. in HUA to the method for theorem 6.3. in HUA (see [23]) and the reformulation presented in chapter 3 that includes the symbol mapping $\{b \mapsto c\}$.

Theory Term Mapping

The meta-method `Theory Term Mapping` replaces certain *reference terms* at all occurrences within a method M. The intended goal is to change the actually given

[5] An `Extended Symbol Mapping` can try to equalize the assumptions as well.

representation such that concl(M') becomes equal to the conclusion thm of the given problem $(ass; thm)$[6].

The meta-method is as follows:

Metamethod: `Theory Term Mapping`	
parameter	P: problem
pre	P=$(ass; thm)$ and there exists a mapping f:{terms}$\mapsto${terms} such that concl(M)[$term_i$/f($term_i$)]$_i$=thm, where $term_i$ are the reference terms, and for a given background theory T we have $T \vdash term_i$=f($term_i$) for all i
post	M'=M[$term_i$/f($term_i$)]$_i$
procedure	PROCTERM-TH (see below)
rating	TERM-rating

Here $term_i(x_1, \ldots, x_n)$ are the reference terms mapped by f. We also assume that we have heuristics that select a minimal f out of the possible ones. `Theory Term Mapping` can be justified by equations $term_i$=f($term_i$) in the theory T. The existence of a theory that justifies the replacement influences TERM-rating. Of course the problem is to find a mapping f: {terms} $\mapsto$ {terms} as stated above and also to control the potential proliferation of such mappings. This problem is solved, e.g., by term rewriting systems.

The meta-method `Theory Term Mapping` modifies the preconditions, the postconditions, and the declarative content of a method M by
PROCTERM-TH:

- Replace all occurrences of the reference terms $term_i(t_1, \ldots, t_n))$ in M by the image terms f($term_i(t_1, \ldots, t_n)$), where $term_i(t_1, \ldots, t_n)$ is an instance of $term_i$.
- Modify the corresponding quantification and sort declarations if necessary, i.e., if affected by the term replacement.

Term Mapping

The meta-method `Term Mapping` also replaces certain reference terms $term_i$ and their instances respectively by other terms f($term_i$) and their instances at all occurrences in M. The purpose is to change the actually given method to M' such that concl(M')=thm for a given problem $(ass; thm)$. As a heuristic restriction, that avoids search explosion, the reference terms have to be maximal terms of concl(M). That means that concl(M) and thm should have the same logical structure. Another constraint which at the same time serves as a justification for `Term Mapping` requires

[6] `Theory Term Mapping` could be complemented by an `Extended Term Mapping` whose goal is to match the assumptions of the methods as well.

that ass(M)$[term_i/\mathrm{f}(term_i)]_i \subseteq ass\cup$ KB.

Examples

The transformation of the proof of 6.9. in HUA to a proof of 13.7. in HUA by `Term Mapping` is successfully established with the mapping

$$\mathrm{f}(term) = \begin{cases} \lambda(x, yz) \cdot \lambda(y, z) : term = \Phi(x(\Phi(y, z))) \\ \lambda_0(x, yz) \cdot \lambda(y, z) : term = \Phi_0(x, \Phi(y, z)) \end{cases}$$

The transformation of the proof of theorem 17.6. in HUA to an analogous proof was based on the mapping f: $\mathrm{f}(x_1 \cdot x_2) = (x_2 \cdot x_1)$.

Dualities

Dualities are common in mathematics, e.g., the Stone Duality between (objects in) topological spaces and (maximal ideals in) Boolean Algebras. They are good justifications for a meta-method that replaces a formula occurring in a method M by its dual. This correspondence shows how an interpretation of a theory T1 – by mapping the formulae in T1 to formulae in T2 – provides another theory T2, some axioms or theorems of which are images of the axioms of T1. This concept is applied in IMPS [15] where the interpretations (i.e., the mappings that establish the duality) are given by the user.

Add Argument

This meta-method changes a unary function to a binary one. It should be applied if concl(M) of the M_{1*}-method M equals the conclusion of an M_{2*}-method after replacing the unary function symbol f by a binary function symbol f'. `Add Argument` is applicable if the unary function symbol f occurs in the conclusion of M.

Metamethod: `Add Argument`	
parameter	P: problem
pre	P=$(ass; thm)$ and term $\mathtt{f}(x)$ occurs in concl(M) and thm=concl(M)$[\mathtt{f}(x)/\mathtt{f}'(x,y)]$
post	M'=M$[\mathtt{f}(t_1)/\mathtt{f}'(t_1,t_2)]$, where t_1, t_2, are terms
procedure	PROCADD (see below)
rating	ADD-rating

The symbols `f, f'` are the function variables to be instantiated.

Terms of the form $\mathtt{f}(term1)$ that occur in M, where $term1$ is a term, are the *reference terms*. PROCADD replaces `f` by `f'` in the parameter slot and modifies the preconditions, the postconditions, and dec-cont(M) as follows:

- Replace in M all occurrences of reference terms $\mathtt{f}(term1)$ by $\mathtt{f}'(term1, term2)$ where $term2$ is obtained from $term1$ by a syntactic replacement of all variables x_{ij} and constants c_{jk} by new variables y_{ij} and new constants d_{jk} of the same sorts respectively. Also modify the quantification accordingly.
  Example: Replace $f(x)$ in a formula
  $\phi_1(f(x)) = f(\phi_1(x))$ by $f'(x, y)$ and $f(\phi_1(x))$ by $f'(\phi_1(x), \phi_1(y))$ with the result $\phi_1(f'(x, y)) = f'(\phi_1(x), \phi_1(y))$.

- Replace in all those formulae which were affected by the previous replacement the occurrences of subformulae
  $(\forall)\phi(x_{i1}, \ldots, x_{ik}, c_{j1}, \ldots, c_{jl})$, which are maximal with respect to `not f` (i.e., the maximal term not containing `f`), by
  $(\forall)(\phi(x_{i1}, \ldots, x_{ik}, c_{j1}, \ldots, c_{jl}) \wedge \phi(y_{i1}, \ldots, y_{ik}, d_{j1}, \ldots, d_{jl}))$, where x_{ij} and c_{jk} are variables and constants of the reference terms and y_{ij} and d_{jk} are the replacing variables and constants of above.

  For instance, $\forall x(\phi_1(c_0) = x \wedge f(\phi_1(c_0)) = f(x))$ is replaced by
  $\forall x, y(\phi_1(c_0) = x \wedge \phi_1(d_0) = y \wedge f'(\phi_1(c_0), \phi_1(d_0)) = f'(x, y))$

- If some of the constants c, which occur in a reference term, also occur in a line l_c of dec-cont(M), but no reference term is in this line, a copy l_d of l_c has to be inserted into dec-cont(M). This copy l_d is obtained from l_c by the same renaming of the constants occurring in the reference terms as above.

  Then insert a line l_{cd} that contains the conjunctions of the formulae in l_c and l_d, the method ($\wedge$ I), and support lines l_c, l_d. Also, replace in all dec-cont lines the support line name l_c by l_{cd}.

  For instance: $(l_c \ldots \vdash \forall x(c_0 \in S \wedge \phi_1(c_0) = x \wedge \phi(c_0) \in H) \ldots)$, insert l_d and l_{cd}
  $(l_d \ldots \vdash \forall y(d_0 \in S \wedge \phi_1(d_0) = y \wedge \phi(d_0) \in H) \ldots)$, and
  $(l_{cd} \ldots \vdash \forall x(c_0 \in S \wedge \phi_1(c_0) = x \wedge \phi(c_0) \in H) \wedge$
  $\quad \forall y(d_0 \in S \wedge \phi_1(d_0) = y \wedge \phi(d_0) \in H)(\wedge I l_c, l_d))$.

`Add-Argument` can be generalized to n-ary functions f or applied recursively.

Example

`Add-Argument` is, for example, used for the reformulation of theorem 7.5.7. to theorem 5.7. in HUA where this application of `Add Argument` essentially maps the function variable Op from $Op(x)$ to $Op(x_1, x_2)$. For more details see chapter 5.

Replace Assumptions

This meta-method reformulates the assumptions of an M_{1*}-method M, if its conclusion already equals the conclusion of an M_{2*}-method and the assumptions of M and the M_{2*}-method differ in few elements only. The postcondition of M_{2*} is $\mathsf{P}{=}(ass; thm)$.

`Replace Assumptions` is presented here for the case ass(M)=$\Delta \cup \{\phi_1\}$ and $ass = \Delta \cup \{\phi_2, \phi_3\}$.
(It can be extended to other cases such as ass(M)=$\Delta \cup \{\phi_1, \phi_2\}$ and $ass = \Delta \cup \{\phi_3\}$ or ass(M)=$\Delta \cup \{\phi_1\}$ and $ass = \Delta \cup \{\phi_2\}$.)

As this meta-method changes $\Delta \cup \{\phi_1\}$ to $\Delta \cup \{\phi_2, \phi_3\}$, it has to change all the lines of dec-cont(M) that refer in any way to ϕ_1.

The scheme of `Replace Assumptions` looks as follows:

Metamethod: `Replace Assumptions`	
parameter	P: problem
pre	P=$(ass; thm) \wedge$ concl(M)=$thm \wedge$ for some $\Delta, \phi_1, \phi_2, \phi_3$ ass(M)=$\Delta \cup \{\phi_1\} \wedge ass = \Delta \cup \{\phi_2, \phi_3\} \wedge \phi_1 \notin \Delta$
post	ass(M')=$\Delta \cup \{\phi_2, \phi_3\}$
procedure	PROCRA (see below)
rating	RA-rating

The procedure PROCRA modifies ass(M) and dec-cont(M) as follows:

- Replace in ass(M) ϕ_1 by ϕ_2, ϕ_3.
- Replace the line (l_n. $\{\phi_1\} \vdash \phi_1$ (HYP)) by lines
  (l_{n1}. $\{\phi_2\} \vdash \phi_2$ (HYP)) and
  (l_{n2}. $\{\phi_3\} \vdash \phi_3$ (HYP)) and ($l_{n'}$. $\{\phi_2, \phi_3\} \vdash \phi_2 \wedge \phi_3$ ($\wedge I\ \ l_{n1}, l_{n2}$)).
- Replace each line of the form (l_i. $\Sigma \cup \{\phi_1\} \vdash \Phi$ (M$_i$ $l_n \ldots$)) by a line
  ($l_{i'}$. $\Sigma \cup \{\phi_2, \phi_3\} \vdash \Phi$ (PLAN$_i$ $l_{n'} \ldots$)).

Tautological Equivalence

This meta-method makes the postconditions of two methods compatible by a propositional reformulation. More specifically, for a method M and a problem P=$(ass; thm)$ the formulae from ass(M) and concl(M) are replaced by tautologically equivalent formulae from ass and thm.

Adjust Definitions

If `Handle Definitions` has been applied, concl(M) has not been affected . Therefore superficial differences of the conclusion of an M$_{1*}$- and an M$_{2*}$-method still exist which are differences between expanded definitions and corresponding formulae. `Handle Definitions` applied to a method M and a problem P tries to make these differences disappear in that it generates conclusions of M and P which are either both expanded definitions or both formulae are of the form $P(c_1, \ldots, c_n)$.

4.1.3 Meta-methods for Abstraction

Some of the most important reformulation techniques in theorem proving are generalization and abstraction. According to Kodratoff and Michalski generalizations are reformulations along the subset dimension, i.e., every model of a formula is a model of the generalized formula. Abstractions are reformulations that eliminate the information which is not relevant for the goal [24]. According to this view of abstraction, our abstracting meta-methods should preserve the important proof steps and the important parts of the formulae and eliminate others. In practice, however, many of the known reformulations are generalizations as well as abstractions.

For analogy-driven theorem proving, only those abstractions are of interest which are restricted by the requirement[7]:
If abstraction(problem$_1$)=abstraction(problem$_2$),
then reverse_abstraction(abstraction(proof(problem$_1$)))
is an outline for a proof of problem$_2$.

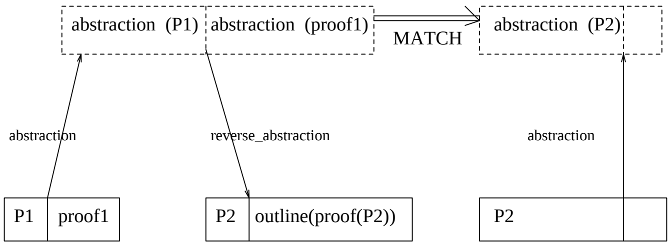


Figure 7: Abstraction for analogy

All abstractions presented in this paper are essentially term mappings. They have, however, different preconditions which must be fulfilled. Characteristics of meta-methods for abstraction are: They are applied to M_{1*}-methods and M_{2*}-methods, they fill the history slot of M, and they delete information from dec-cont(M).

Abstraction of Homomorphism

A homomorphism F is a mapping $S \Rightarrow T$ that is characterized essentially by a formula of the form
$\forall x_1, \ldots, x_n(F(f_S(x_1, \ldots, x_n)) = f_T(F(x_1), \ldots, F(x_n)))$, where f_S, f_T are functions in S and T respectively. However, the defining formulae may differ from this standard case, e.g., if S and T are H-semimoduls as in many cases of HUA. Then the defining formula becomes

[7] Giunchiglia and Walsh [17] speak of a different kind of abstraction (PI) if a proof of the abstracted problem exists and provides an outline of the proofs of the un-abstracted problems. Our abstracted proofs need not be proofs of the abstracted problem.

$\forall x, y(x \in S \wedge y \in H \to F(f_S(y,x)) = f_T(y, F(x))$, i.e., y is not mapped by the homomorphism F.

The abstraction of such a defining formula describes a homomorphism as a mapping respecting any operation on the structure S. This is achieved by the meta-method `Hom-Abstr` that replaces concrete function symbols by variables, abstracts certain terms in the defining formula and drops variables which are not mapped by the homomorphism.

This meta-method can be applied to a method M if ass(M) contains the definition Φ of a homomorphism F, which has the general form
$hom\_from\_S(F) \leftrightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m(x_1, \ldots, x_n \in S \wedge \psi(y_1, \ldots, y_m) \to$
$F(term(x_1, \ldots, x_n)) = term'(F(x_1), \ldots, F(x_n)))$,

where the $x_1, \ldots, x_n$ are exactly the variables in S, i.e., the variables which are affected by the mapping F. $\psi(y_1, \ldots, y_m)$ is a conjunction of formulae that contain the variables y_i only. These conjunctive parts of ψ are called the *superfluous formulae*.

The reference terms $term, term'$ occurring in the defining formula Φ are abstracted throughout M.

The abstraction affects all instances of the reference terms `term` and `term'` in that it replaces them everywhere in the method M. The superfluous formulae are omitted by the abstraction. For instance, a quantifier and the formula $(f \in F)$ becomes superfluous if
$hom\_from\_S(\phi) \leftrightarrow \forall x, f(x \in S \wedge f \in F \to \phi(f \cdot x) = f \cdot \phi(x))$
is abstracted to
$hom\_from\_S(\phi) \leftrightarrow \forall x(x \in S \to \phi(Op(x)) = Op'(\phi(x)))$.

Metamethod: `Hom-Abstr`	
parameter	P: problem
pre	P=$(ass; thm)$ and there exists a formula $\phi =$ $hom\_from\_\mathtt{S}(\mathtt{F}) \leftrightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m(x_1, \ldots, x_n \in \mathtt{S} \wedge$ $\psi(y_1, \ldots, y_m) \to$ $\mathtt{F}(\mathtt{term}(x_1, \ldots, x_n)) = \mathtt{term}'(F(x_1), \ldots, F(x_n)))$ and $\phi \in$ ass(M) and $\phi \notin ass$, where `term, term'` are terms.
post	M'=M[$\mathtt{term}(t_1 \ldots t_n)/Op(t_1 \ldots t_n)$, `term'`$(t_1 \ldots t_n)/Op'(t_1 \ldots t_n)$]
procedure	PROCHOM (see below)
rating	HOM-rating

`F` is a function variable, `S` a variable for a structure, and `term, term'` are terms where the instances of `term` and `term'` differ in their function symbols only.

(Hom: $hom\_from\_\mathtt{S}(\mathtt{F}) \leftrightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m(x_1, \ldots, x_n \in \mathtt{S} \wedge \psi(y_1, \ldots, y_m) \to$ $\mathtt{F}(\mathtt{term}(x_1, \ldots, x_n)) = \mathtt{term}'(F(x_1), \ldots, F(x_n))))$ is stored in the history slot of M' in order to make a later reversion of the abstraction possible.

The procedure PROCHOM modifies the slots of M by executing the following:

- Check ass(M) for defining formulae of a homomorphism, such as, for instance, $hom\_from\_S(\phi_1) \leftrightarrow \forall x, f(x \in S \wedge f \in F \rightarrow \phi_1(f \cdot_S x) = f \cdot_T \phi_1(x))$

- Replace all occurrences of the instances of the reference terms $\texttt{term}(t_1 \ldots t_n)$ in M by $Op(t_1, \ldots, t_n)$ and $\texttt{term'}(t_1 \ldots t_n)$ in M by $Op'(t_1, \ldots, t_n)$ (where $t_1, \ldots, t_n$ are terms) with the new function variables Op, Op'.

- Drop the superfluous formulae and superfluous quantifiers of variables that occurred within $\texttt{term}(t_1 \ldots t_n)$, but do not occur in $Op(t_1, \ldots, t_n)$, where `term` is a reference term.

- Add (Hom:$hom\_from\_\texttt{S}(\texttt{F}) \leftrightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m(x_1, \ldots, x_n \in \texttt{S} \wedge \psi(y_1, \ldots, y_m) \rightarrow \texttt{F}(\texttt{term}(x_1, \ldots, x_n)) = \texttt{term}'(F(x_1), \ldots, F(x_n))))$ to the history slot.

- Add the function variables Op, Op' to the parameter slot.

Example
`Hom-Abstr` was successfully applied to submethods within the treatment of theorems 5.7. and 7.5.7. in HUA (see chapter 5).

Functional Abstraction

Under certain conditions this meta-method abstracts a term (the reference term) with a free variable x to a term $f_a(x)$, where f_a is a new function variable. It also drops quantifiers and membership declarations of variables y_i that are no longer relevant, where conjunctions of formulae of the form $x_i \in S_i$ for some set S_i are denoted as **membership declarations**. The preconditions for the application of `Functional Abstraction` is that there is a formula Φ in post(M) that has the form
$\forall x_1, \ldots, x_n, y_1 \ldots y_k$ (membership decl. $\rightarrow \phi(x_1 \ldots, x_n) \rightarrow \phi(\texttt{term}(x_1), \ldots, \texttt{term}(x_n))$),
where the variables y_i occur in `term` only. This formula should not occur in problem P.

An example of such a formula is
$\forall y, x_1, x_2(x \in F \wedge x_1, x_2 \in F \rightarrow (R(x_1, x_2) \rightarrow R(y \cdot x_1, y \cdot x_2))))$
which is abstracted to
$\forall x_1, x_2(x_1, x_2 \in F \rightarrow (R(x_1, x_2) \rightarrow R(f_a(x_1), f_a(x_2))))$.

Metamethod: `Functional-Abstr`	
parameter	P: problem
pre	there exists a formula Φ of the form $\forall x_1, \ldots, x_n, y_1 \ldots y_k$ (membership declaration $\rightarrow$ $\phi(x_1 \ldots, x_n) \rightarrow \phi(\mathtt{term}(x_1), \ldots, \mathtt{term}(x_n)))$ and Φ in post(M) and Φ not in P
post	M'=M[$\mathtt{term}(x_i)/f_a(x_i)]_i$
procedure	PROCFUNC(see below)
rating	FUNC-rating

The x_i are the maximal terms in $\phi(x_1 \ldots, x_n)$. `term` is a metavariable for the *reference term* that contains only one variable.
`Functional-Abstr` reformulates a method M to a method M' by executing the procedure PROCFUNC:

- Replace in M all occurrences of instances of the reference term $\mathtt{term}(t_i)$ by $f_a(t_i)$, where f_a is a new function variable.
- Delete membership declarations that became superfluous by the introduction of f_a, and delete the corresponding quantifiers.

  For example, if `Functional-Abst` replaces the reference term $(h \cdot x)$ by $f_a(x)$, then the quantifier and membership declaration of $h, (h \in F)$ become superfluous.
- Add (Functional-Abst: $\forall x_1, \ldots, x_n, y_1 \ldots y_k$(membership decl. $\rightarrow \phi(x_1 \ldots, x_n)$ $\rightarrow \phi(\mathtt{term}(x_1), \ldots, \mathtt{term}(x_n))))$ to the history slot of M.
- Add the new parameter f_a to the parameter slot of M.

Example
`Functional-Abstraction` can be applied among others to a method that encodes a subproblem of 5.2. of HUA and its proof. The assumptions of the method contain the formulae
$\Phi = \forall h, f_1, f_2(h, f_1, f_2 \in F \rightarrow R(f_1, f_2) \rightarrow R(h \cdot f_1, h \cdot f_2))$.
Hence, $h \cdot t$ is replaced by $f_a(t)$ for terms t.

The assumptions of the analogue of subproblem 5.2.1. of HUA contain the formula
$\Phi = \forall h, f_1, f_2(h \in F \wedge f_1, f_2 \in S \rightarrow R(f_1, f_2) \rightarrow R(h \cdot f_1, h \cdot f_2))$.
Hence, $h \cdot t$ is replaced by $f_a(t)$ for terms t.

List Abstraction

Under certain conditions this meta-method abstracts a list (the *reference list*) to a term $\mathrm{f}_l([x_1, \ldots, x_n])$, where f_l is a new function variable, and drops membership declarations of variables which do no longer occur in the postconditions.

The preconditions for the application of `List Abstraction` assert that a formula Φ should occur in post(M) but not in P, where Φ is of the form $\forall x_1, \ldots, x_n, y_1, \ldots, y_m$(membership decl.$\rightarrow \phi_1([x_1, \ldots, x_n]) \rightarrow \phi_2(\mathtt{list}(x_1, \ldots, x_n))))$. $[x_1, \ldots, x_n]$ denotes the list which is the maximal term in $\phi_1([x_1, \ldots, x_n])$ and $\mathtt{list}(x_1, \ldots, x_n)$ denotes the reference list that is the maximal term in $\phi_2(\mathtt{list}(x_1, \ldots, x_n))$ the elements of which are terms that are dependent on some of the $x_1, \ldots, x_n$. An example for such a formula Φ is the definition of $symmetric(R)$: $\forall x_1, x_2([x_1, x_2] \in R \rightarrow [x_2, x_1] \in R)$, where $\mathtt{list}(x_1, x_2) = [x_2, x_1]$ .The terms, i.e. here just the two variables, x_2 and x_1 are dependent on $\{x_2, x_1\}$.

The meta-method looks schematically as follows:

Metamethod: `List-Abstr`	
parameter	P: problem
pre	P$=(ass, concl) \wedge \Phi = \forall x_1, \ldots, x_n, y_1 \ldots y_k$(membership declarations $\rightarrow \phi_1([x_1, \ldots, x_n]) \rightarrow \phi_2(\mathtt{list}(x_1, \ldots, x_n)) \wedge$ $\Phi \notin ass \wedge \Phi \in$ ass(M)
post	M'=M($\mathtt{list}(t_1, \ldots, t_n)/\mathtt{f}_l([x_1, \ldots, x_n])$)
procedure	PROCLIST (see below)
rating	LIST-rating

The denotations are explained above. The variables y_i of Φ may occur in the part $\mathtt{list}(x_1, \ldots, x_n)$ only. PROLIST modifies the slots of M by executing:

- Replace $\mathtt{list}(x_1, \ldots, x_n)$ by $\mathtt{f}_l([x_1, \ldots, x_n])$ with the new function variable $\mathtt{f}_l$ and delete the quantifiers and membership declarations which became superfluous.

- Replace $\mathtt{list}(t_1, \ldots, t_n)$ by $\mathtt{f}_l([t_1, \ldots, t_n])$ in any instantiation of Φ where the x_i are instantiated by terms t_i. Drop the superflous quantifiers and the membership declarations which became superfluous.

- Replace the instances of reference lists everywhere in M and drop the superfluous quantifiers andmembership declarations.

- Add (`List-Abstr`: $\forall x_1, \ldots, x_n, y_1 \ldots y_k$(membership decl.$\rightarrow \phi_1([x_1, \ldots, x_n]) \rightarrow \phi_2(\mathtt{list}(x_1, \ldots, x_n)))$ to the history slot of M.

- Add a new parameter $\mathtt{f}_l$ to the parameter slot.

Example

The meta-method `List-Abstr` was applied in the transformation of a subproof of theorem 4.8. to a subproof of theorem 5.3. of HUA with the reference lists $[x_2, x_1]$ and $[gx_1, gx_2]$ respectively. The respective theorems are:

Theorem 4.8 Let ρ and σ be two equivalence relations, then
1. $(\rho \cap \sigma)$ is an equivalence relation and
2. $(\rho \cup \sigma)^t$ is the smallest equivalence relation, containing ρ and σ.

Theorem 5.3 Let ρ and σ be two equivalence relations, then
1. $(\rho \cap \sigma)$ is a leftcongruence and
2. $(\rho \cup \sigma)^t$ is the smallest leftcongruence containing ρ and σ.

By structuring the method (4.8.1.2) for the problem $(\ldots \vdash symm((\rho \cap \sigma))$ and the method (5.3.1.b) for the problem
$(\ldots \vdash (x, y) \in (\rho \cap \sigma) \rightarrow (fx, gy) \in (\rho \cap \sigma))$ arise, respectively. After normalization ass(4.8.1.2) contains the expanded definition of $symm$
$\forall x_1, x_2((x_1, x_2) \in \rho \rightarrow (x_2, x_1) \in \rho))$ which is not in ass(5.3.1.b), and ass(5.3.1.b) contains the expanded definition of $leftcongruence$
$\forall g, x_1, x_2(g \in H \rightarrow (x_1, x_2) \in \rho \rightarrow \in \rho(gx_1, gx_2) \in \rho))$ which is not in ass(4.8.1.2). Hence, `List-Abstr` is applicable to both normalized methods and the postconditions of the resulting new method match.

Please note that the usual symbol mapping in work on analogy would be $symm \mapsto leftcongruence$, and this would results in concl(4.8.1.2) = concl(5.3.1.b). However it does not provide a match of the respective method assumptions since they contain the definitions of $symm$ and $leftcongruence$ respectively. Even an additional term mapping would still not yield the required matching.

4.1.4 Meta-methods for Restructuring

In this subsection we present some changes of representation obtained by restructuring. Some of the meta-methods correspond to well-known methods (e.g., conjunctive decomposition). However, the meta-methods have another (meta-theoretic) purpose, namely to expose the structure of a proof/a method.

Although the presented meta-methods were sufficient for our examples, more elaborated meta-methods have to be developed for this class (see below).

The structuring meta-methods eventually change the proof plan structure by splitting a method into several new methods or they combine existing methods to a new method. Structuring meta-methods can be applied to M_{1*}- and M_{2*}-methods. They do not change the history slot. Their purpose is to create methods for subproblems, i.e., they are necessary for establishing analogies of subproblems.

Conjunctive Decomposition

This meta-method is applied in order to split a conjunctive conclusion into two conclusions. `Conjunctive Decomposition` is applicable to an argument method M if concl(M) has the form $(\forall)(A \wedge B)$. It changes M to M' and generates two new methods M1, M2 with

- concl(M1)=$(\forall)A$ and concl(M2)=$(\forall)B$ respectively, and pre(M = pre(M1) = pre(M2).
- Depending on the method occurring in the line ($l.\ ass \vdash A \wedge B$(METHOD ...)), dec-cont(M1) and dec-cont(M2) are constructed.
  - If METHOD equals PLAN, then generate M1 with ass(M1)=(ass(M)$\cup A$) or if there is a dec-cont(M)-line $(\ldots ass1 \vdash A \ldots)$ for some $ass1 \subseteq ass$ then ass(M1)=$(ass1 \cup A)$; with pre(M1)=pre(M).

    Take the lines of dec-cont(M) for dec-cont(M1) except for the lines ($l.\ ass \vdash A \wedge B$)(PLAN) and ($l_2.\ \Delta \vdash B \ldots$) where $\Delta \subseteq ass$. Add the line ($l_1.\ ass \vdash A$ (PLAN1)) to dec-cont(M1) if dec-cont(M) does not contain a line $(\ldots \Delta \vdash A \ldots)$ with $\Delta \subseteq ass$. Handle M2 analogously.
  - If METHOD equals $(\wedge I)$, that is, there are lines l_1, l_2 $(l_1.\ ass1 \vdash A \quad (M_i, \ldots))$ and $(l_2.\ ass2 \vdash B \quad (M_j \ldots))$, then generate a method M1 with ass(M1)=$ass1$ and concl(M1)=A, pre(M1)=pre(M). In dec-cont(M1) delete the lines $(l.\ ass \vdash A \wedge B \quad (\wedge I, l_1, l_2))$ and l_2. Proceed analogously for M2.
- dec-cont(M) is changed to
  $(l_1.\ ass \vdash A \quad \text{(LEMMA)})$
  $(l_2.\ ass \vdash B \quad \text{(LEMMA)})$
  $(l_3.\ ass \vdash (A \wedge B) \quad (\wedge I, l_1, l_2))$.

  pre(M) is augmented by $(ass \vdash A)$ and $(ass \vdash B)$.

Equivalence Decomposition

If concl(M)=$(\forall)(A \leftrightarrow B)$, then the meta-method splits M into submethods M_1 and M_2 with
concl(M_1)=$(\forall)(A \rightarrow B)$ and for concl(M_2)=$(\forall)(B \rightarrow A)$.
This meta-method was applied to the method of theorem 5.2. in HUA.

Other suggestions for structuring

Additional structuring heuristics have been investigated, inter alia by:

- Bishop Brock et al. [3] propose to introduce so-called motivations for proof parts such that the range of one motivation represents a subproof.

- Bishop Brock et al. [3] and Stephen Owen [27] suggest heuristics for identifying the key steps in proofs. Then the proof parts leading from one key step to another can be proposed as subproofs. The application of certain assumptions, such as definitions, were identified as key steps of a proof. A step where temporarily introduced symbols are removed, often represents the completion of a part of a proof. Hence, the step immediately following the removal is usually a key step.

- If a problem is used several times in a proof, the derivation of this problem could be considered as a subproof.

- Proofs incorporating an application of the Deduction Theorem can be split into one kernel part and another part that consists of the actual application of the Deduction Theorem. (Its precondition is concl(M) = $((\forall)A \rightarrow B)$; if $A \vdash B$ occurs in dec-cont(M) or M is an initial method, then M is split; else a new method is created by adding a line with $A \vdash B$ to dec-cont(M).)

- An obvious strategy is to split a method M according to the occurrence of names of non-basic methods in dec-cont(M). This meta-method is important in particular if the base method is structured already.

4.2 Other Meta-methods

Now we collect some additional meta-methods which we have used.

Plan Realization

This meta-method is applied in VERIFICATION (see chapter 3). `Plan Realization` removes method variables from dec-cont(M), which is a condition for a verification of M.

More specifically, if an intermediate (i.e., not the last line) PLAN-line $(\ldots \Delta \vdash \phi(PLAN))$ occurs in dec-cont(M), then `Plan Realization` creates a method M' differing from M by the additional precondition $(\Delta; \phi)$. It changes the PLAN-line to a LEMMA-line in dec-cont(M). During the completion of the proof plan this new precondition causes an additional method M_i to be inserted, preceding M', with post(M_i) = $(\Delta; \phi)$.

Change Plan

This meta-method should be applied before `Plan Realization`. The purpose is to change or to simplify PLAN-lines of dec-cont(M), if one knows that if ϕ_2 could be proved then ϕ is provable. `Change Plan` finds a precondition ϕ_1 for the proof of a formula ϕ which is to be proved by a PLAN-line. Then the actual proof of ϕ is no longer unknown but it is derived from the formula ϕ_1, the proof of which is unknown. More specifically, the meta-method looks as follows:

Metamethod: `Change-Plan`	
parameter	
pre	There exist lines ($l_i. \ldots \vdash \phi$ (PLAN)) and ($l_k. \ldots \vdash \phi_1 \ldots$) and there is a formula $\Phi = \phi_1 \wedge \phi_2 \rightarrow \phi and$ $\Phi \in$ (ass(M) $\cup$ KB) and $(\phi_1 \rightarrow \phi) \notin$ (ass(M) $\cup$ KB)
post	ass(M') = ass(M) $\cup\{\Phi\}$, new PLAN-line (see below)
procedure	PROCCH (see below)
rating	CH-rating

PROCCH modifies the slots of M in that it

- adds $\Phi = \phi_1 \wedge \phi_2 \rightarrow \phi$ to ass(M),
- adds the lines ($l_m.\ \{\Phi\} \vdash \Phi\ (HYP)$),
  ($l_o.\ ass_o \vdash \phi_2$ (PLAN)), and
  ($l_n.\ ass_o \cup ass_k \vdash \phi_1 \wedge \phi_2\ (\wedge I, l_k, l_o)$) to dec-cont(M),
- replaces the line ($l_i. \ldots \vdash \phi$ (PLAN)) by ($l_i. \ldots \vdash \phi\ (\rightarrow D, l_m, l_n)$).

The resulting method M' looks schematically as follows:

Method: `M'`	
parameter	same as in M
pre	same as in M
post	ass(M')=ass(M) $\cup\{concl(3)\}$
dec-cont	1. ; $\vdash \phi_2$ (PLAN) 2. ; $\vdash \phi_1$ (same as in M) 3. 3; $\vdash \phi_1 \wedge \phi_2 \rightarrow \phi$ (HYP) 4. 1, 3; $\vdash \phi_1 \wedge \phi_2$ ($\wedge I$ 3 1) 5. 3, 4; $\vdash \phi$ ($\rightarrow D$ 3 4) 6. ; $\vdash$ the remaining lines of dec-cont(M) (same as in M)
procedure	schema interpreter
history	same as in M

`Change-Plan` is our version of Bledsoe's precondition prover PC [2]. However, Bledsoe's procedure is more complicated and has additional features.

This meta-method has been used to prove theorem 13.7. of HUA, analogously to the proof of theorem 6.9. in HUA.

Reversion

There are several variants of `Reversion` because different history slot fillers may occur as parameters of the meta-method. During the analogy-driven plan construction meta-methods from the Reversion class reformulate M_{1*}-methods, the postcondition of which has matched the postcondition of an abstracted or normalized M_{2*}-method in such a way that the new postcondition matches with the original P2.

The parameters are the entries of the history slot of the M_{2*}-method.

4.3 Control Strategies

An analogy between two theorems and their proofs can be established in many different ways. Our process of analogy-establishing REFORMULATION starts first with the current methods M_1 and M_2, constructed in INITIALIZE, the postconditions of which do not yet match. Several M_{1*}- and M_{2*}-methods are computed by REFORMULATION and added to the set of current methods which are tried for matching or for further reformulation. The goal of the reformulation is then to obtain postconditions of the M_{1*}-method and M_{2*}-method that match. If that is impossible we try to find subproblems that match.

At any point in time during the reformulation process there may be several meta-methods applicable to more than one method, hence the need for control strategies.

A first and important control strategy fixes the right choice of the class of reformulations and these classes are to be activated in a fixed sequence; afterwards we have to pick the heuristically best choice within each class.

The general sequence of these classes that turned out to be most useful is:

1. Normalization
2. Abstraction
3. Direct Reformulation
4. Restructuring

If the meta-methods in Structuring generate new methods, then the reformulation cycle starts again with Normalization. The above sequence is realized in the algorithm CONSTRUCT_PROOF_BY_ANALOGY by the function NEXT.

- **NORMALIZATION**
  Try to apply meta-methods from the normalization class to the current M_{1*}-nad M_{2*}-methods as long as possible. This produces new current M_{1*}- and M_{2*}-methods .

  Characterization:
  Normalizing meta-methods try to make the postconditions of methods compatible. They are applicable to M_{1*}- *and* M_{2*}-methods. They fill the history

slot of methods. They do not provide new parameters for the method. They do not delete information from the method.

- **ABSTRACTION**
  Apply abstracting meta-methods to the current M_{1*}- and M_{2*}-methods as long as possible.

  Characterization:
  These meta-methods fill the history slot of the argument methods. They introduce new parameters, and they delete information from the declarative content.

- **DIRECT REFORMULATION**
  Try to apply directly reformulating meta-methods to the current M_{1*}-methods as long as possible with the aim to match the conclusions of corresponding M_{1*}- and M_{2*}-methods. This is dependent on the conclusion of the M_{2*}-method the reformulation is directed to.

  Characterization:
  These meta-methods only change M_{1*}-methods. They do not change the history slot.

- **RESTRUCTURING**
  Finally, try to apply meta-methods that restructure the current M_{1*}- and M_{2*}-methods. In particular, the splitting meta-methods are of interest. The resulting submethods (together with the included subproblems) are the new methods to be treated by the analogy-driven plan construction. Restructuring serves the special purpose to reveal the structure of a proof/a method within the reformulation process.

  Characterization:
  Structuring meta-methods can be applied to M_{1*}- and M_{2*}-methods. They change the number of methods to be considered for a plan.

Within each of these classes there are many meta-methods that could be applied and their choice is taken by a planning procedure that uses information from the rating slot of meta-methods and heuristics. This planner, in a simple case the user, decides which of the meta-methods from the active class of meta-methods is to be applied in the current situation. The actual design of the planner is outside the scope of this paper.

A control strategy for the right choice of methods to be considered for a reformulation is needed as well. There is often little reason to prefer one method over another initially. Therefore an agenda-based control as proposed in [27] and [3] turned out to be helpful. The methods organized in the agenda are ordered by a heuristic measure of their promise. One of the heuristics to be employed is: Less abstract methods should be chosen reformulation before more abstract methods.

5 Examples

5.1 The Extended Example of de la Tour and Kreitz

This example was already discussed in chapter 3, we now repeat it in more technical detail.
We start with method M_1 that looks as follows:

Method: $\mathtt{M}_1$	
parameter	
pre	
post	$\emptyset; \forall x, y((Qxy \to \forall z(Pxz \to Pyz)) \wedge Qab \wedge (Paa \vee Pbb)) \to \exists x Pbx)$
dec-cont	1. ; 1 $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz)) \wedge Qab \wedge (Paa \vee Pbb)$ (HYP) 2. ; 1 $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz))$ ($\wedge D$ 1) 3. ; 1 $\vdash (Paa \vee Pbb)$ ($\wedge D$ 1) ——— the proof, case 1 ——— 4. 4; $\vdash Pbb$ (HYP) 5. 4; $\vdash \exists x Pbx$ ($\exists I$ 4) ——— case 2 ——— 6. 6; $\vdash Paa$ (HYP) 7. ; 1 $\vdash Qab$ ($\wedge D$ 1) 8. ; 1 $\vdash \forall z(Paz \to Pbz)$ ($\forall D, \to D$ 1 7) 9. 6; 1 $\vdash Pba$ ($\forall D, \to D$ 8 6) 10. 6; 1 $\vdash \exists x Pbx$ ($\exists I$ 9) 11. ; 1 $\vdash \exists x Pbx$ ($\vee D$ 10 5 3) 12. ; $\vdash \forall x, y((Qxy \to \forall z(Pxz \to Pyz)) \wedge Qab \wedge (Paa \vee Pbb) \to \exists x Pbx)$ ($\to I$ 11)
procedure	schema-interpreter
history	

The method M_2 is:

Method: $\mathtt{M}_2$	
parameter	
pre	
post	$\emptyset; \forall x, y((Qxy \to \forall z(Pxz \to Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \to \exists x Pcx)$
dec-cont	1. ; $\vdash \forall x, y((Qxy \to \forall z(Pxz \to Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc) \to \exists x Pcx)$ (PLAN)
procedure	schema-interpreter
history	

M_1 is normalized by the meta-method `Deduction Normal-Form` to M_{11}:

Method: `M`$_{11}$	
parameter	
pre	
post	$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, (Paa \vee Pbb)\}$; $\exists x Pbx$
dec-cont	1. ; 1 $\vdash$ $\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz))$ (HYP) 2. ; 2 $\vdash$ $(Paa \vee Pbb)$ (HYP) — the proof,case 1 — 3. 3; $\vdash$ Pbb (HYP) 4. 3; $\vdash$ $\exists x Pbx$ ($\exists I$ 3) — case 2 — 5. 5; $\vdash$ Paa (HYP) 6. ; 6 $\vdash$ Qab (HYP) 7. ; 1, 6, 2 $\vdash$ $\forall z(Paz \rightarrow Pbz)$ ($\forall D, \rightarrow D$ 1 6) 8. 5; 1, 6, 2 $\vdash$ Pba ($\forall D, \rightarrow D$ 7 5) 9. 5; 1, 6, 2 $\vdash$ $\exists x Pbx$ ($\exists I$ 8) 10. ; 1, 6, 2 $\vdash$ $\exists x Pbx$ ($\vee D$ 2 4 9)
procedure	schema-interpreter
history	Ded: 1(a), a. and Conj; $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb)) \rightarrow \exists x Pbx$

Note that the hypotheses introduced in lines 1, 2, and 6 are assumptions. The hypotheses introduced in lines 3 and 5 actually are hypotheses which have to be removed later on in the proof.

M_2 is normalized to M_{21} by the meta-method: `Deduction Normal-Form` to:

Method: `M`$_{21}$	
parameter	
pre	
post	$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, Qbc, (Paa \vee Pcc)\}$; $\exists x Pcx$
dec-cont	1. ; 1 $\vdash$ $\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz))$ (HYP) 2. ; 2 $\vdash$ Qab (HYP) 3. ; 3 $\vdash$ Qbc (HYP) 4. ; 4 $\vdash$ $(Paa \vee Pcc)$ (HYP) 5. ; 1, 2, 3, 4 $\vdash$ $\exists x Pcx$ (PLAN)
procedure	schema-interpreter
history	Ded: 1(b), a, and Conj: $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge Qbc \wedge (Paa \vee Pcc)) \rightarrow \exists x Pcx$

Then M_{11} is reformulated by `Symbol Mapping` $(b \mapsto c)$ to M_{12}:

Method: $\mathtt{M}_{12}$	
parameter	
pre	
post	$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qac, (Paa \vee Pcc)\}$; $\exists x Pcx$
dec-cont	1. ; 1 $\vdash$ $\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz))$ (HYP) 2. ; 2 $\vdash$ $(Paa \vee Pcc)$ (HYP) — the proof,case 1 — 3. 3; $\vdash$ Pcc (HYP) 4. 3; $\vdash$ $\exists x Pcx$ ($\exists I$ 3) — case 2 — 5. 5; $\vdash$ Paa (HYP) 6. ; 6 $\vdash$ Qac (HYP) 7. ; 1, 6, 2 $\vdash$ $\forall z(Paz \rightarrow Pcz)$ ($\forall D, \rightarrow D$ 6 1) 8. 5; 1, 6, 2 $\vdash$ Pca ($\forall D, \rightarrow D$ 5 7) 9. 5; 1, 6, 2 $\vdash$ $\exists x Pcx$ ($\exists I$ 8) 10. ; 1, 6, 2 $\vdash$ $\exists x Pcx$ ($\vee D$ 2 4 9)
procedure	schema-interpreter
history	Ded: 1(a), a, and Conj; $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb)) \rightarrow \exists x Pbx$

M_{12} is reformulated by `Replace Assumptions` to M_{13}:

Method: $\mathtt{M}_{13}$	
parameter	
pre	
post	$\{\forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)), Qab, Qbc, (Paa \vee Pcc)\}; \exists x Pcx)$
dec-cont	1. ; 1 $\vdash \forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz))$ (HYP) 2. ; 2 $\vdash (Paa \vee Pcc)$ (HYP) — the proof,case 1 — 3. 3; $\vdash Pcc$ (HYP) 4. 3; $\vdash \exists x Pcx$ ($\exists I$ 3) — case 2 — 5. 5; $\vdash Paa$ (HYP) 6. ; 6 $\vdash Qab$ (HYP) 7. ; 7 $\vdash Qbc$ (HYP) 8. ; 6, 7 $\vdash Qab \wedge Qbc$ ($\wedge I$ 6 7) 9. ; 1, 6, 2, 7 $\vdash \forall z(Paz \rightarrow Pcz)$ (PLAN 1 8) 10. 5; 1, 6, 2, 7 $\vdash Pca$ ($\forall D, \rightarrow D$ 9 5) 11. 5; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\exists I$ 10) 12. ; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\vee D$ 2 4 11)
procedure	schema-interpreter
history	Ded: 1(a), a, and Conj: $\forall x, y((Qxy \rightarrow \forall z(Pxz \rightarrow Pyz)) \wedge Qab \wedge (Paa \vee Pbb)) \rightarrow \exists x Pbx$

The assumptions and conclusions of M_{13} match those of M_{21}. Hence, REVERSION and VERIFICATION can be applied to this method. `Reversion` reformulates M_{13} to M_{14} on the basis of the information in the history slot of M_{21}:

Method: $\mathtt{M}_{14}$	
parameter	
pre	
post	$\emptyset; \forall x, y(Qxy \to \forall z(Pxz \to Pyz) \wedge Qab \wedge Qbc \wedge (Paa \vee Pbb) \to \exists x Pcx$
dec-cont	1. ; 1 $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz))$ (HYP) 2. ; 2 $\vdash (Paa \vee Pcc)$ (HYP) — the proof,case 1 — 3. 3; $\vdash Pcc$ (HYP) 4. 3; $\vdash \exists x Pcx$ ($\exists I$ 3) — case 2 — 5. 5; $\vdash Paa$ (HYP) 6. ; 6 $\vdash Qab$ (HYP) 7. ; 7 $\vdash Qbc$ (HYP) 8. ; 6, 7 $\vdash Qab \wedge Qbc$ ($\wedge I$ 6 7) 9. ; 1, 6, 2, 7 $\vdash \forall z(Paz \to Pcz)$ (PLAN) 10. 5; 1, 6, 2, 7 $\vdash Pca$ ($\forall D, \to D$ 5 9) 11. 5; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\exists I$ 10) 12. ; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\vee D$ 2 4 11) 13. ; $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz) \wedge Qab \wedge Qbc \wedge (Paa \vee Pbb) \to \exists x Pcx$ ($\to I$ 12)
procedure	schema-interpreter
history	

In order to obtain a verifiable method, the method variable PLAN is removed by `Plan Realization`. It reformulates M_{14} to M_{15}, which can now be verified.

Method: $\mathtt{M}_{15}$	
parameter	
pre	$\{concl(1), concl(2), concl(6), concl(7)\}, \forall z(Paz \to Pcz)$
post	$\emptyset; \forall x, y(Qxy \to \forall z(Pxz \to Pyz)) \land Qab \land Qbc \land$ $(Paa \lor Pbb) \to \exists x Pcx$
dec-cont	1. ; 1 $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz))$ (HYP) 2. ; 2 $\vdash (Paa \lor Pcc)$ (HYP) —— the proof,case 1 —— 3. 3; $\vdash Pcc$ (HYP) 4. 3; $\vdash \exists x Pcx$ ($\exists I$ 3) —— case 2 —— 5. 5; $\vdash Paa$ (HYP) 6. ; 6 $\vdash Qab$ (HYP) 7. ; 7 $\vdash Qbc$ (HYP) 8. ; 6, 7 $\vdash Qab \land Qbc$ ($\land I$ 6 7) 9. ; 1, 6, 2, 7 $\vdash \forall z(Paz \to Pcz)$ (LEMMA) 10. 5; 1, 6, 2, 7 $\vdash Pca$ ($\forall D, \to D$ 9 5) 11. 5; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\exists I$ 10) 12. ; 1, 6, 2, 7 $\vdash \exists x Pcx$ ($\lor D$ 2 4 11) 13. ; $\vdash \forall x, y(Qxy \to \forall z(Pxz \to Pyz)) \land Qab \land Qbc \land (Paa \lor Pbb) \to \exists x Pbx$ ($\to I$ 12)
procedure	schema-interpreter
history	

M_2 is now replaced by M_{15}.

In order to obtain a proof plan for P2 as complete as possible that incorporates M_{15}, a new method M_{2a} has to be introduced that succeeds M_{15} in the plan. Its postcondition has to be equal to pre(M_{15}) and its precondition has to be empty. Such a method to fill the gap between pre(M_{15}) and $\emptyset$ can be, for instance,

Method: $\mathtt{M}_{2a}$	
parameter	
pre	
post	$\{concl(1), concl(2), concl(3)\}; \forall z(Paz \rightarrow Pcz)$
dec-cont	1. ; 1 $\vdash \forall x, y(Qxy \rightarrow \forall z(Pxz \rightarrow Pyz))$ (HYP) 2. ; 2 $\vdash Qab$ (HYP) 3. ; 3 $\vdash Qbc$ (HYP) 4. ; 1, 2 $\vdash (Pad \rightarrow Pbd)$ $(\forall D, \rightarrow D, \forall D\ 1\ 2)$ 5. ; 1, 3 $\vdash (Pbd \rightarrow Pcd)$ $(\forall D, \rightarrow D, \forall D\ 1\ 3)$ 6. 6; $\vdash Pad$ (HYP) 7. 6; 1, 2 $\vdash Pbd$ $(\forall D, \rightarrow D\ 4\ 6)$ 8. 6; 1, 2, 3 $\vdash Pcd$ $(\rightarrow D\ 7\ 5)$ 9. ; 1, 2, 3 $\vdash Pad \rightarrow Pcd$ $(\rightarrow I\ 8)$
procedure	schema-interpreter
history	

To summarize, the analogy-driven proof plan construction consists of the following sequence of steps:

1. (a) Applying `Deduction Normal-Form` to M_1 yields M_{11} and applying `Deduction Normal-Form` to M_2 yields M_{21}.
   (b) `Symbol Mapping` $\{y_0 \mapsto x_0, x_0 \mapsto y_0\}$ applied to M_{11} provides M_{12}.
   (c) `Replace Assumptions` reformulates M_{12} to M_{13} and introduces a PLAN-line in dec-cont(M_{13}). The postcondition of M_{13} equals the postcondition of M_{21}.
   (d) M_{13} is reformulated to M_{14} by `Reversion`.
   (e) `Plan Realization` applied to M_{14} yields M_{15}.
2. M_{15} can be verified, and M_2 is replaced by M_{15}.
3. Plan Planning inserts an additional method M_{2a} with concl(M_{2a})= $\forall z(Paz \rightarrow Pcz)$ and ass(M_{2a}) $\subseteq$ ass(M_2).

5.2 Examples from HUA

Essentially the same reformulation steps as before can be applied to the first part of a subproof of theorem 17.6. of HUA such that we get an analogous second part of this subproof. These two parts of the subproof arise by unfolding definitions and subsequent structuring. The subtheorem at hand is of the form $(A \leftrightarrow B)$ and the second part $(\leftarrow)$ is to be proven analogously to the first part $(\rightarrow)$ of the subproof.

Such a structure of a proof is typical for certain analogies in mathematical theorem proving:

The first subproblem is:
$\{leftcongr(R), agreeable(R,E)\}; \forall R, x, y((x,y) \in R \rightarrow \forall h(h \in F \wedge hx \in E \rightarrow hy \in E))$.

The second subproblem is:
$\{leftcongr(R), agreeable(R,E)\}; \forall x, y((x,y) \in R \rightarrow \forall h(h \in F \wedge hy \in E \rightarrow hx \in E))$.

Now let us look at another example from HUA:
The task is to prove subtheorem 5.7.2.c of 5.7. of HUA by analogy to theorem 7.5.7.2. The actual theorems are,(translated into English):

Theorem 5.7.2.c:
If S', H_1, H_2 are semigroups and $\phi_1 : S' \Rightarrow H_1, \phi_2 : S' \Rightarrow H_2$ are two homomorphisms into the semigroups H_1 and H_2, respectively, and ρ_1, ρ_2 are the induced congruences respectively, and $\rho_1 \subset \rho_2$, and ϕ_1 is surjective, and there is a $\Phi : H_1 \Rightarrow H_2$ with $\Phi\phi_1 = \phi_2$, then
$\forall x, y(x \in H_1 \wedge y \in H_1 \rightarrow \Phi(x \cdot y) = \Phi(x) \cdot \Phi(y))$.

Theorem 7.5.7.2.c:
Let S, T_1, T_2 be F-semimoduls, and let $\phi_1 : S \Rightarrow T_1$ and $\phi_2 : S \Rightarrow T_2$ be two homomorphisms into the F-semimoduls T_1 and T_2, respectively, and let ρ_1, ρ_2 be the induced leftcongruences respectively. If $\rho_1 \subset \rho_2$ and ϕ_1 is surjective, and there is a $\Phi : H_1 \Rightarrow H_2$ with $\Phi\phi_1 = \phi_2$, then
$\forall f, x(x \in T_1 \wedge f \in F \rightarrow \Phi(f \cdot x) = f \cdot \Phi(x))$.

At an abstract level, both theorems say that Φ is a homomorphism. However the actual definitions of homomorphy differ and, hence, the assumptions, the theorems, and the proofs differ. These differences do not disappear by a symbol map. Hence known analogy approaches fail. Even if one could find a match of the theorems based on tricks such as mapping symbols dependent of their position or argument pairings, one still can not produce certain lines of the proof of 5.7.2.c. from lines of the proof of 7.5.7.2.c. For instance, in former approaches the *two* lines
$(\ldots \exists y : S(\phi_1(y) = b) \ldots)$ and
$(\ldots \exists z : S(\phi_1(y) = c) \ldots)$ of the proof of 5.7.2.c are not obtained from the corresponding line
$(\ldots \exists y : S(\phi_1(y) = b) \ldots)$ of the proof of 7.5.7.2.c.(There are more lines of the proof that one cannot obtain by symbol mapping (see [23]).

As the two theorems are subtheorems of 5.7. and 7.5.7 in HUA (see [23]), a structuring has taken place before and is not included in this presentation for simplicity. The proof plan for the theorem 5.7. as a whole is not constructed either.

Besides, a formulation of the theorems (e.g., of the definitions) has been chosen that actually not requires normalization. Therefore we left out the normalizations (e.g., `Handle Definitions`) as well as the reversion of normalizations. Note: This is done *only* for clearness.

The starting methods are method(5.7.2.c) and method(7.5.7.2.c). For simplicity, let the function symbol in `term` be polymorphic.

<table>
<tr><td colspan="2">Method: <code>7.5.7.2c</code></td></tr>
<tr><td>parameter</td><td></td></tr>
<tr><td>pre</td><td></td></tr>
<tr><td>post</td><td>$\{concl(1), \ldots, concl(15)\}$;
$\forall f, x(f \in F \wedge x \in T_1 \rightarrow \Phi(f \cdot x) = f \cdot \Phi(x))$</td></tr>
<tr><td>dec-cont</td><td>
1. ; 1 $\vdash$ $\forall x, y, f(x, y \in T_2 \wedge f \in F \rightarrow$ $x = y \rightarrow f \cdot x = f \cdot y)$ (KB-HYP)
2. ; 2 $\vdash$ $\forall x, f(x \in S \wedge f \in F \rightarrow f \cdot x \in S)$ (KB-HYP)
3. ; 3 $\vdash$ $\forall x, y, f(x, y \in T_1 \wedge f \in F \rightarrow (x = y \rightarrow$ $f \cdot x = f \cdot y)$ (KB-HYP)
4. ; 4 $\vdash$ hom-from-$S(\phi_1) \leftrightarrow$ $\forall f, x(f \in F \wedge x \in S \rightarrow \phi_1(f \cdot x) = f \cdot \phi_1(x))$ (DEF HYP)
5. ; 5 $\vdash$ hom-from-$T_1(\Phi) \leftrightarrow$ $\forall f, x(f \in F \wedge x \in T_1 \rightarrow \Phi(f \cdot x) = f \cdot \Phi(x))$ (DEF HYP)
6. ; 6 $\vdash$ $\forall x, f(x \in T_1 \wedge f \in F \rightarrow f \cdot x \in T_1)$ (KB HYP)
$\vdots$ $\vdots$ $\vdots$ $\vdots$
15.;15 $\vdash$ $concl(15)$ (HYP)
Proof
$\vdots$ $\vdots$ $\vdots$ $\vdots$
20.;... $\vdash$ $\exists y(y \in S \wedge \phi_1(y) = b)$ (...)
$\vdots$ $\vdots$ $\vdots$ $\vdots$
40.; $\vdash$ $f \in F \wedge b \in T_1 \rightarrow \Phi(f \cdot b) = f \cdot \Phi(b)$ (...)
$\vdots$ $\vdots$ $\vdots$ $\vdots$
70.;1-15 $\vdash$ $\forall f, x(f \in F \wedge x \in T_1 \rightarrow \Phi(f \cdot x) = f \cdot \Phi(x))$ ($\forall I$ 35)
</td></tr>
<tr><td>procedure</td><td>schema-interpreter</td></tr>
<tr><td>history</td><td></td></tr>
</table>

The declarative content of method 7.5.7.2c is partially omitted, to keep this concise, the full proof is presented in [23]. The initial method of theorem 5.7.2.c looks as follows.

Method: `5.7.2c`	
parameter	
pre	
post	$\{concl(1), \ldots, concl(15)\}$; $\forall x_1, x_2(x_1, x_2 \in H_1 \rightarrow \Phi(x_1 \cdot x_2) = \Phi(x_1) \cdot \Phi(x_2))$
dec-cont	1. ; 1 $\vdash$ $\forall x_1, x_2, y_1, y_2(x_1, x_2, y_1, y_2 \in H_2 \rightarrow$ $x_1 = x_2 \wedge y_1 = y_2 \rightarrow x_1 \cdot y_1 = x_2 \cdot y_2)$ (KB-HYP) 2. ; 2 $\vdash$ $\forall x_1, x_2(x_1, x_2 \in S' \rightarrow x_1 \cdot x_2 \in S')$ (KB-HYP) 3. ; 3 $\vdash$ $\forall x_1, x_2, y_1, y_2(x_1, x_2, y_1, y_2 \in H_1 \rightarrow$ $x_1 = x_2 \wedge y_1 = y_2 \rightarrow x_1 \cdot y_1 = x_2 \cdot y_2)$ (KB-HYP) 4. ; 4 $\vdash$ hom-from-S'(ϕ_1) $\leftrightarrow$ $\forall x, y(x, y \in S' \rightarrow \phi_1(x \cdot y) = \phi_1(x) \cdot \phi_1(y))$ (DEF HYP) 5. ; 5 $\vdash$ hom-from-$H_1(\Phi)$ $\leftrightarrow$ $\forall x, y(x, y \in H_1 \rightarrow \Phi(x \cdot y) = \Phi(x) \cdot \Phi(y))$ (DEF HYP) 6. ; 6 $\vdash$ $\forall x, y(x, y \in H_1 \rightarrow x \cdot y \in H_1)$ (KB HYP) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 15. ;15 $\vdash$ $concl(15)$ (HYP) ——— Proof ——— 16. ; $\vdash$ $\forall x, y(x, y \in H_1 \rightarrow \Phi(x \cdot y) = \Phi(x) \cdot \Phi(y))$ (PLAN)
procedure	schema-interpreter
history	

Since the definition of hom-from-S is in ass(7.5.7.2c), `Hom-Abst` can be applied with the key term $term(x) = (f \cdot x)$. The application of `Hom-Abstr` to the method (7.5.7.2.c) yields the method (7.5.7.2cA) by the transformation $(f \cdot x) \mapsto Op_1(x)$ and by omitting superfluous set declarations and quantifiers.

The method(5.7.2c) is abstracted by `Hom-Abstr` with the key term $term(x, y) = (x \cdot y)$ using the transformation $(x \cdot y) \mapsto Op_2(x, y)$ for the method 5.7.2cA.

<table>
<tr><td colspan="2">Method: <code>5.7.2cA</code></td></tr>
<tr><td>parameter</td><td>Op_2: function</td></tr>
<tr><td>pre</td><td></td></tr>
<tr><td>post</td><td>$\{concl(1), \ldots, concl(15)\}; \forall x_1, x_2 (x_1, x_2 \in H_1 \rightarrow \Phi(Op_2(x_1, x_2)) = Op_2(\Phi(x_1), \Phi(x_2)))$</td></tr>
<tr><td>dec-cont</td><td>
1. ; 1 $\vdash$ $\forall x_1, x_2, y_1, y_2 (x_1, x_2, y_1, y_2 \in H_2 \rightarrow x_1 = x_2 \wedge y_1 = y_2 \rightarrow Op_2(x_1, y_1) = Op_2(x_2, y_2))$ (KB-HYP)
2. ; 2 $\vdash$ $\forall x_1, x_2 (x_1, x_2 \in S' \rightarrow Op_2(x_1, x_2) \in S')$ (KB-HYP)
3. ; 3 $\vdash$ $\forall x_1, x_2, y_1, y_2 (x_1, x_2, y_1, y_2 \in H_1 \rightarrow x_1 = x_2 \wedge y_1 = y_2 \rightarrow Op_2(x_1, y_1) = Op_2(x_2, y_2))$ (KB-HYP)
4. ; 4 $\vdash$ hom-from-S'(ϕ_1) $\leftrightarrow \forall x, y (x, y \in S' \rightarrow \phi_1(Op_2(x, y)) = Op_2(\phi_1(x), \phi_1(y)))$ (DEF HYP)
5. ; 5 $\vdash$ hom-from-$H_1(\Phi) \leftrightarrow \forall x, y (x, y \in H_1 \rightarrow \Phi(Op_2(x, y)) = Op_2(\Phi(x), \Phi(y)))$ (DEF HYP)
6. ; 6 $\vdash$ $\forall x, y (x, y \in H_1 \rightarrow Op_2(x, y) \in H_1)$ (KB HYP)
$\vdots$ $\vdots$ $\vdots$ $\vdots$
15. ;15 $\vdash$ $concl(15)$ (HYP)
abstracted proof
16. ; $\vdash$ $\forall x, y (x, y \in H_1 \rightarrow \Phi(Op_2(x, y)) = Op_2(\Phi(x), \Phi(y)))$ (PLAN;)
</td></tr>
<tr><td>procedure</td><td>schema-interpreter</td></tr>
<tr><td>history</td><td></td></tr>
</table>

Method: `7.5.7.2cA`	
parameter	Op_1: function
pre	
post	$\{concl(1), \ldots, concl(15)\}; \forall x(x \in T_1 \rightarrow \Phi(Op_1(x)) = Op_1(\Phi(x)))$
dec-cont	1. ; 1 $\vdash$ $\forall x, y(x, y \in T_2 \rightarrow x = y \rightarrow Op_1(x) = Op_1(y))$ (KB-HYP) 2. ; 2 $\vdash$ $\forall x(x \in S \rightarrow Op_1(x) \in S)$ (KB-HYP) 3. ; 3 $\vdash$ $\forall x, y(x, y \in T_1 \rightarrow x = y \rightarrow Op_1(x) = Op_1(y))$ (KB-HYP) 4. ; 4 $\vdash$ hom-from-S(ϕ) $\leftrightarrow$ $\forall x(x \in S \rightarrow \phi(Op_1(x)) = Op_1(\phi(x)))$ (DEF-HYP) 5. ; 5 $\vdash$ hom-from-T_1(Φ) $\leftrightarrow$ $\forall x(x \in T_1 \rightarrow \Phi(Op_1(x)) = Op_1(\Phi(x)))$ (DEF-HYP) 6. ; 6 $\vdash$ $\forall x(x \in T_1 \rightarrow Op_1(x) \in T_1)$ (KB-HYP) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 15. ; $\vdash$ $concl(15)$ (HYP;) ——— abstracted proof ——— $\vdots$ $\vdots$ $\vdots$ $\vdots$ 20. ;... $\vdash$ $\exists y(y \in S \rightarrow \phi_1(y) = b)$ (...) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 40. ; $\vdash$ $b \in T_1 \rightarrow \Phi(Op_1(b)) = Op_1(\Phi(b))$ (...) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 70. ;1-15 $\vdash$ $\forall x(x \in T_1 \rightarrow \Phi(Op_1(x)) = Op_1(\Phi(x)))$ ($\forall I \ldots$)
procedure	schema-interpreter
history	(Hom-Abst:$f \cdot x \Rightarrow Op_1(x)$)

The postconditions of the methods (5.7.2.cA) and (7.5.7.2.cA) still do not match. Therefore we try a reformulation from the class of direct reformulations. The application of `Add Argument` to (7.5.7.2.cA) yields a method (7.5.7.2.cA') the postconditions of which match the postconditions of (5.7.2.cA).

Method: `7.5.7.2cA'`	
parameter	Op_2: function
pre	
post	$\{concl(1), \ldots, concl(15)\}; \forall x_1, x_2(x_1, x_2 \in T_1 \rightarrow \Phi(Op_2(x_1, x_2)) = Op_2(\Phi(x_1), \Phi(x_2)))$
dec-cont	1. ; 1 $\vdash \forall x_1, x_2, y_1, y_2(x_1, x_2, y_1, y_2 \in T_2 \rightarrow x_1 = x_2 \wedge y_1 = y_2 \rightarrow Op_2(x_1, x_2) = Op_2(y_1, y_2))$ (HYP) 2. ; 2 $\vdash \forall x_1, x_2(x_1, x_2 \in S \rightarrow Op_2(x_1, x_2) \in S)$ (HYP) 3. ; 3 $\vdash \forall x_1, x_2, y_1, y_2(x_1, x_2, y_1, y_2 \in T_1 \rightarrow x_1 = x_2 \wedge y_1 = y_2 \rightarrow Op_2(x_1, x_2) = Op_2(y_1, y_2))$ (HYP) 4. ; 4 $\vdash$ hom-from-S(ϕ) $\leftrightarrow \forall x_1, x_2(x_1, x_2 \in S \rightarrow \phi(Op_2(x_1, x_2)) = Op_2(\phi(x_1), \phi(x_2)))$ (DEF HYP) 5. ; 5 $\vdash$ hom-from-T_1(Φ) $\leftrightarrow \forall x_1, x_2(x_1, x_2 \in T_1 \rightarrow \Phi(Op_2(x_1, x_2)) = Op_2(\Phi(x_1), \Phi(x_2)))$ (DEF HYP) 6. ; 6 $\vdash \forall x_1, x_2(x_1, x_2 \in T_1 \rightarrow Op_2(x_1, x_2) \in T_1)$ (KB HYP) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 15. ;15 $\vdash$ $concl(15)$ (HYP) ——— Reformulation(abstracted proof) ——— $\vdots$ $\vdots$ $\vdots$ $\vdots$ 20. ;... $\vdash \exists y(y \in S \rightarrow \phi_1(y) = b)$ (...) 21. ;... $\vdash \exists z(z \in S \rightarrow \phi_1(y) = c)$ (...) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 40. ;... $\vdash b \in T_1 \wedge c \in T_1 \rightarrow \Phi(Op_2(b, c)) = Op_2(\Phi(b), \Phi(c))$ (...) $\vdots$ $\vdots$ $\vdots$ $\vdots$ 88. ; $\vdash \forall x_1, x_2(x_1, x_2 \in T_1 \rightarrow \Phi(Op_2(x_1, x_2)) = Op_2(\Phi(x_1), \Phi(x_2))$ ($\forall$ I ...)
procedure	schema-interpreter
history	Hom-Abst:$f \cdot x \Rightarrow Op_1(x)$

The postcondition parameter as well as the proof is reformulated by `Add Argument` . For instance, a line
$(\ldots \vdash \exists y(y \in S \wedge \phi_1(y) = b) \ldots)$ from the main part of the proof schema of (7.5.7.2cA) is replaced by two lines
$(\ldots \vdash \exists y(y \in S \wedge \phi_1(y) = b) \ldots)$ and
$(\ldots \vdash \exists z(z \in S \wedge \phi_1(y) = c) \ldots)$,
and a line
$(\ldots \vdash b \in T_1 \rightarrow \Phi(Op_1(b)) = Op_1(\Phi(b)) \ldots)$ is replaced by a line
$(\ldots \vdash b \in T_1 \wedge c \in T_1 \rightarrow \Phi(Op_2(b, c)) = Op_2(\Phi(b), \Phi(c)) \ldots)$.

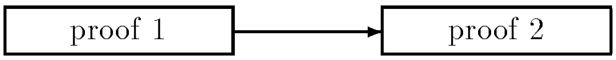
In order to obtain a method whose postconditions match the postconditions of (5.7.2.c) the former abstraction of (5.7.2.c) has to be made undone by `Reversion`. This results in (7.5.7.2cA"). Since there is no PLAN-line in the resulting method, we try to verify method (7.5.7.2.cA") without further preparation. The only method available for constructing a proof plan for the problem 5.7.2.c is (7.5.7.2cA"). We do not need to find additional methods for filling gaps as method (7.5.7.2cA") has no preconditions.

6 Conclusion

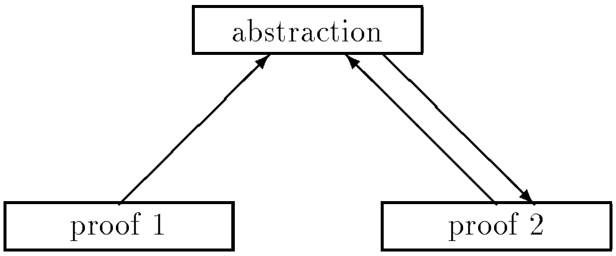
The framework proposed in this paper is based on the described techniques for reformulation and on the proof plan methodology, such that the construction of proofs for analogous problems becomes in principle independent of the actual problem representation.

Within this framework several types of analogies can be established:

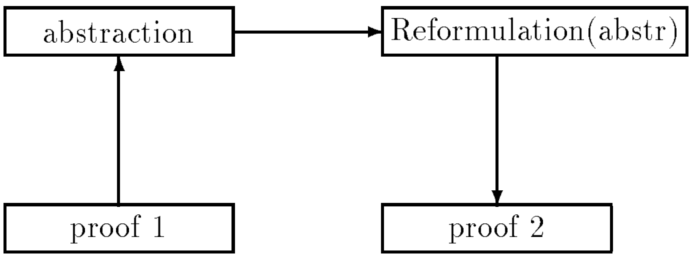
- Analogies based on direct mapping of proofs onto proofs, as in previous approaches to theorem proving by analogy:



- Analogies based on abstractions of proofs (and subsequent reverse_abstraction):



- Analogies based on abstracted and in addition reformulated proofs with subsequent reverse_abstraction.



To summarize the main points of our approach:

1. Theorems may be given by various representations and this entails that the commonality of two problems is often very implicit only. A *reformulation* of the representation is hence necessary to identify the similarity of such problems.

   According to this view of analogous problems, the computer supported reformulation becomes an important component of analogy-driven theorem proving.

2. Reformulation includes normalization, structuring, and abstraction.

   - Normalization removes some superficial differences between the base and the target problem. These changes are hardly ever explicitly mentioned by mathematicians, but they have to be explicitly coped with by a computer. Such reformulations have not been taken into account in former approaches to analogy, which presupposed that there are no such differences between the base and the target problem.
   - Structuring is a powerful and important means in mathematical theorem proving, in proof planning and in particular in analogy-driven proof plan construction: If an appropriate reformulation of a method can not be found, then the reformulation of a submethod which is produced by structuring, is often useful.
   - Abstraction was not included in previous approaches although it plays an important rôle in proofs by analogy in many cases.

3. The approach is tied into the proof planning methodology by taking *methods* as our basic units too. We take advantage of the following features of methods:

   - Methods can represent *proof ideas* as they may have PLAN-lines and they doubt have to be fully instantiated.

     Methods can explicitly represent the structure of a proof by combining several simple methods to more complex chunks that build the parts of the structure. If the base method is structured this way, it represents a proof idea for the given theorem, which may be easier to transfer than a fully instantiated proof. Thus analogies of proof ideas can be obtained by reformulating such methods into other methods.

     Methods need not be fully instantiated (although most of our base methods are). Because of this characteristic, methods can represent methods in the usual mathematical sense such as, for example, the diagonalization method. Hence the very same method can represent several proofs or parts of proofs which are analogous to each other.
   - Because of the common representation of a problem and its proof in one method, the theorem *and* its proof changes if the method is reformulated. These changes are no longer separated as in previous approaches

and much of the subsequent repair of the target proof caused by changing the problem can be avoided by an adequate reformulation.

However many open problems remain:

- Is the set of presented meta-methods complete in the sense of covering all important reformulations in mathematics?

  Of course not, and in fact, one cannot expect a complete set of meta-methods. More meta-methods have to be created for other mathematical fields.

- Have the presented meta-methods been designed specifically to handle the particular examples considered here?

  The suggested meta-methods have been constructed for particular methods such that all the problems of a standard textbook could be handled. As noted in the previous section, these meta-methods can be used for other purposes and other mathematical fields as well. For example, some of the meta-methods presented here have played a role in tasks outside of analogy-driven proof construction (see e.g., `Add Argument` in [19]).

- If many more meta-methods will be invented by other researchers, and if all then have to be stored for use in a system, how can we cope with the resulting large meta-method search space?

  This is of course a problem, however by carefully choosing only those meta-methods relevant for a mathematical subfield, the search space will be reasonably bounded. Such a classification and additional control information would enable the planner to choose meta-methods in a goal-directed manner.

- Is the method language appropriate?

  Basically, the object language of the methods could be extended, although it was sufficient for our examples, since at the moment we did not handle less instantiated proofs.

The main incentive for future work, however, will undoubtedly come from the experimentation on additional cases from other mathematical fields.

Acknowledgement

I am grateful to Jörg Siekmann who encouraged me for a long time to work on this paper and read and improved previous versions of it. I appreciate Dieter Hutter's discussions on drafts of this paper. Thanks to Manfred Kerber and Xiaorong Huang who have also read earlier drafts of this paper.

References

[1] D. Basin and T. Walsh. Difference matching. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes of Computer Science*, pages 295–309. Springer, 1992.

[2] W.W. Bledsoe. A precondition prover for analogy. Memo ATP 102, Computer Science Department University of Texas, Austin, TX, February 1992.

[3] B. Brock, S. Cooper, and W. Pierce. Some experiments with analogy in proof discovery. Tech.Rep. AI-347-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.

[4] B. Brook, S. Cooper, and W. Pierce. Analogical reasoning and proof discovery. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, pages 454–468, Argonne, 1988. Springer.

[5] A. Bundy. The use of explicit plans to to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120, Argonne, 1988. Springer.

[6] A. Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991.

[7] 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.

[8] J.G. Carbonell and M. Veloso. Integrating derivational problem solving architecture. In J. Kolodner, editor, *Proc. of a workshop on Case-Based Reasoning*, pages 104–124, Florida, 1988.

[9] J. Cleve and D. Hutter. A new methodology for equational reasoning. Technical report, Universität des Saarlandes, 1993.

[10] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W.Harper, D.J. Howe, T.B. Knoblock andN.P. Mendler, P.Paranangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, N.J., 1986.

[11] T.R. Davis and S.J. Russell. A logical approach to reasoning by analogy. In *Proceedings of the Tenth Internaitonal Joint Conference on Artificial Intelligence*, pages 264–270, Milan Italy, 1987. Morgan Kaufmann.

[12] T. Boy de la Tour and Ch. Kreitz. Building proofs by analogy via the curry-howard isomorphism. In A. Voronkov, editor, *Proceedings of Logic Programming and Automated Reasoning '92*, volume 624 of *Lecture Notes on Artificial Intelligence*, pages 202–213, St. Petersburg, 1992. Springer Verlag.

[13] P. Deussen. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher*. Springer, 1971.

[14] G. Faltings and U. Deker. Interview: Die Neugier, etwas ganz genau wissen zu wollen. *bild der wissenschaft*, (10):169–182, 1983.

[15] W. Farmer, J. Guttmann, and J Thayer. Little theories. In D. Kapur, editor, *Proc. 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence Science*, pages 567–581. Springer, 1992.

[16] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Math. Zeitschrift*, 39, 1935.

[17] F. Giunchiglia and T. Walsh. Tree subsumption: Reasoning with outlines. In *Proceedings of 11th ECAI*, pages 77–81, Vienna, 1992.

[18] R.P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39(1):39–120, 1989.

[19] X. Huang, M. Kerber, and M. Kohlhase. Methods - the basic units for planning and verifying proofs. SEKI-Report SR-92-20 (SFB), Universität des Saarlandes, 1992.

[20] B. Indurkhya. *Metaphor and Cognition*. Kluwer Academic Publisher, Dordrecht, 1992.

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

[22] D. Loveland. *Automated Theorem Proving: A Logical Basis*. North Holland, New York, 1978.

[23] E. Melis. Analogies between proofs – a case study. SEKI-Report SR-93-, Universität des Saarlandes, 1993.

[24] R. Michalski and Y. Kodratoff. Research in machine learning. In R. Michalski and Y. Kodratoff, editors, *Machine Learning III. An Artificial Intelligence Approach*, pages 3–30. Morgan Kaufmann, San Mateo, 1990.

[25] J.C. Munyer. *Analogy as a Means of Discovery in Problem Solving and Learning*. PhD thesis, University of California, Santa Cruz, 1981.

[26] A. Newell. The Heuristic of George Polya and its Relation to Artificial Intelligence. Technical Report CMU-CS-81-133, Carnegie-Mellon-University, Dept. of Computer Science, Pittsburgh, Pennsylvania, U.S.A., 1981.

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

[28] G. Polya. *Mathematics and Plausible Reasoning.* Priceton University Press, NJ, 1954.

[29] S.J. Russell. *The Use of Knowledge in Analogy and Induction.* Pitman, London, 1989.