Proving a Heine-Borel Theorem by Analogy

Erica Melis*
University of Edinburgh, Department of AI
80 South Bridge, Edinburgh EH1 1HN, Scotland

Abstract

This paper addresses analogy-driven automated theorem proving that employs a source proof-plan to guide the search for a proof-plan of the target problem. The approach presented uses reformulations that go beyond symbol mappings and that incorporate frequently used re-representations and abstractions. Several realistic math examples were successfully processed by our analogy-driven proof-plan construction. One challenge example, a Heine-Borel theorem, is discussed here. For this example the reformulaitons are shown step by step and the modifying actions are demonstrated.

1 Introduction

Analogy in theorem proving has received little attention despite its importance in mathematics and the claims made for its usefulness in theorem proving [Polya, 1957; Bledsoe, 1986; Wos, 1988]. Reasons for this situation are manifold: Firstly, different from simple AI domains, minor changes in theorems or proof assumptions may cause major changes in proofs. Hence, the retrieval of a source problem is more difficult than usually. And, since previous approaches to analogy always tried to transfer *proofs* analogically, those minor changes of the problem caused a break down of the analogy of the proofs or at least imposed unsolved problems on the modification.

Secondly, it is a well known fact that constructing an analogy in mathematics often amounts to first finding the appropriate representation which brings out the similarity of two problems, that is, finding the right concepts and the right level of abstraction. Previous approaches to analogy in theorem proving [Kling, 1971; Munyer, 1981; Owen, 1990], however, used symbol mapping only rather than employing re-representations or abstractions. Hence, their results were highly dependent on the actual representation of the theorems and of proof assumptions.

Thirdly, previous approaches were able to handle only examples that are very simple compared to real maths examples (except Woody Bledsoe and his students in [Brook *et al.*, 1988; Bledsoe, 1995] who worked on debugging analogies). They, hence, failed to produce worked compelling examples.

Our approach [Melis, 1995b] contributes to overcoming this situation by working at the proof-*plan* level, by offering reformulations that correspond to re-representations or abstractions of problems, and by processing real maths examples. This paper first introduces the new approach and presents a challenge example afterwards.

2 Analogy-Driven Proof-Plan Construction

Proof-plans, introduced in [Bundy, 1988], are high level representations of proofs that consist of methods. We postulate that the transfer of proof-plans by using and transforming their methods is an appropriate level of abstraction at which to draw analogies. The power of such an analogical transfer stems from the given proof-plan, from transferring proof strategies encoded into subplans, and from reformulating proof-plans which may include the reformulation of methods.

Besides, often proof-plans are better suited for analogical transfer than formal proofs which are often too brittle to apply a transformation in general. But still, proof-plans contain enough information to construct a concrete proof for a given problem.

As discussed in section 4 our analogy model is general enough to work for different proof planning systems. The methods and reformulations, however, may vary between these systems. Since we shall present the worked example in the planning framework and with the operators designed for Ω-MKRP [Huang *et al.*, 1994a], we start with a brief review of how these operators and plans are defined, introduce reformulation and discuss the analogy procedure.

Operators

Sequents $\mathsf{P} = (\Delta \vdash F)$, are pairs of a set Δ of formulas and a formula F in an object language that is extended by meta-variables for functions, relations, formulas, sets of formulas, and terms.

Proof-plan operators, called *methods*, were first introduced in [Bundy, 1988]. The methods used here are frame-like structures defined in [Huang *et al.*, 1994b].

*On leave from University Saarbrücken, Germany

method: `Indirect Proof`	
parameter	F: formula, Δ: set of formulas
preconditions	$(\Delta \cup \{\neg F\} \vdash \perp)$
postcondition	$(\Delta \vdash F)$
constraints	
proof schema	1. $\neg F \quad \vdash \neg F$ (HYP) 2. $\Delta, \neg F \quad \vdash \perp$ (LEMMA) 3. $\Delta \quad \vdash F$ (IP;2)
procedure	standard schema-interpreter

An example of a method is `Indirect Proof` in which IP is a composite ND rule.

More specifically, methods M have the following slots: parameter, preconditions (pre(M))[1], postcondition (post(M)), constraints, proof schema and procedure. pre(M) is a set of sequents from which the application of the method derives post(M) which is a sequent as well. pre(M) and post(M) both are needed for planning. The constraints are formulated in a meta-language and serve to restrict the search during planning, e.g., restrictions of pre(M), post(M), or of the parameters. The proof schema is a declarative schematic representation of proofs in the object logic, relying on the Natural Deduction (ND) calculus and on invoking an automated theorem prover (atp) such as OTTER [McCune, 1990]. The standard program in the slot procedure executes the application of the proof schema.

Our methods mainly differ from those in [Bundy, 1988] in that the tactic slot is replaced by a declarative proof schema *and* a procedure interpreting this schema[2]. The intention behind this difference is to enable reformulations of methods that change the underlying tactic as well.

A method is *verifiable* if, given pre(M), then the method yields a correct proof of post(M). For verifying a line with an atp-call, a time limit is set for the atp to prove the sequent.

Proof Plans

Since in general maths proofs are constructed top down and bottom up, we consider backward *and* forward search in proof planning and define plan operators to be an f-method or a b-method respectively. f-methods work with their preconditions as input and postcondition as output; b-methods work vice versa. For instance, the method corresponding to the ND-rule $\wedge$-elimination is a typical f-method, whereas the method $\wedge$-introduction is a typical b-method. f- and b-methods will be treated differently by the analogy procedure. (Yes, ND-rules correspond to basic methods in this framework but compound methods are predefined and can be constructed by the user. Essentially, the analogy procedure manipulates a *given* source plan and does not care about the level of methods. As shown in the example, the source proof-plan encodes chunks of proofs.)

Goals and assumptions are sequents, and a *proof-plan* is a forest the trees of which consist of sequent nodes and method nodes that satisfy the "link condition": A method node M follows a sequent node g and precedes the sequent nodes $\mathsf{g}_1, \ldots, \mathsf{g}_n$ if $\sigma(\text{post}(M)) = \mathsf{g}$ and $\sigma(\text{pre}(M)) = \{\mathsf{g}_1, \ldots, \mathsf{g}_n\}$ for a substitution σ of parameters.

Proof planning starts with a goal g and assumptions $(\emptyset \vdash F_i)$, where F_i are proof-assumptions, axioms, definitions, or lemmas. The proof planning proceeds by inserting methods and sequents: A b-method follows a goal and yields new (sub)goals as its successors. An f-method precedes assumptions and yields a new preceding assumption. Planning aims at reducing the gap between leaf goals and assumptions. Leaf goals that are not equal to an assumption are called *open goals*. As soon as a goal g_i equals an assumption, the two nodes collapse. Then g_i is no longer an open goal but *satisfied*. The planning terminates if there are no open goals.

The source proof-plans are trees with the source problem at the root, with no open goals, and with verifiable methods. For the analogy procedure we use *linearised* proof-plans ordered by the sequence in which the nodes have been added to the plan. The analogy follows a *linearised* plan because the sequence of introducing methods can be important for the mapping and reformulation, particularly in case of forward *and* backward planning. As in [Veloso, 1994] justification structures, used to encode justifications for the decision made, annotate the plan nodes. These *justifications* capture the dependency structure of a plan and point to reasons for the choice, such as application conditions of a method[3], user-given guidance, or pre-programmed control knowledge.

2.1 Reformulation

The reformulations included into our analogy, change *proof-plans*. That is, in general they can remove methods, replace a method/subplan by another given method/subplan or by a method/subplan that is constructed by the reformulation. Reformulations are triggered by the aim to match source goals/assumptions with target goals/assumptions and to satisfy justifica-

[1] Note the renaming of slots compared with Bundy's methods.

[2] Besides, the slots are renamed, e.g., our preconditions are named input there.

[3] E.g., its verifiability or the existence of a certain definition

tions. Reformulations are mappings ρ of a proof-plan to a proof-plan which usually but not necessarily preserve the verifiability of methods in a plan. They encode mathematical heuristics on how a proof-plan changes dependent on certain changes of the theorem and assumptions. Reformulations may change the plan, methods, sequents, and justifications of nodes.

The reformulations used in the worked example below are `Symbol-Mapping` and `Add-Arguments` (see, e.g., [Melis, 1995c]). We advocate high-level reformulations and pretty general ones in order to obtain just a small number of reformulations. `Add-Arguments` seems to be general and used in maths more often than not. (Kolbe and Walther [Kolbe and Walther, 1995] discovered a similar procedure in the context of instantiating and patching generalised proofs.)

Reformulations are carried out by meta-methods which are represented by data structures with the slots parameters, application-condition, effect, program. The purpose of the slots application-condition, effect is to meta-plan a sequence of reformulations. program executes the reformulation dependent on parameters.

2.2 Construction

Our analogy-driven proof-plan construction is a derivational analogy (see, e.g., [Carbonell, 1986; Mostow, 1989]). It is a control strategy for proof planning that extends the derivational analogy of [Veloso, 1994] by reformulation and bidirectional planning. The general idea of our analogy model is to use the linearised source proof-plan together with its justifications as a guide for constructing an analogous target proof-plan and to transfer methods (and sequents) of a reformulated source proof-plan to the target proof-plan.

Table 1 shows the top-level procedure of our analogy-driven proof-plan construction. Given a parametrized, linearised source proof-plan, target assumptions, and a target goal (the first open goal), the output of the procedure is a target proof-plan.

Step 4 is relevant for a planner with backward search only. Steps 5-6 cover the transfer of f-methods. The former matches a source goal and transfers a b-method whereas the latter matches as many source assumptions as possible to target assumptions and transfers an f-method. $|missing(\rho'\mathrm{M})| < m$ means[4] that less than m preconditions of the currently treated reformulated f-method M do not match a target assumption. Hence, m is a parameter that expresses the confidence on a success of analogy despite missing target assumptions. The sequents of $missing(\rho'\mathrm{M})$ become new open goals if ρ' is an acceptable reformulation.

The first goal of the linearised source plan, usually the source problem P_S, is chosen in 3. If P_S can be reformulated by a ρ such that it matches the target problem P_T, then ρ will be applied to the (current) source plan. The method M with post(M) $= \rho\mathsf{P}_S$ will be transferred to the target if its justifications hold in the target.

In the example below the check of a method's justifications amounts to the check of its verifiability. The latter test is necessary because reformulations of the presented methods do not necessarily preserve verifiability. If $\rho\mathrm{M}_1$ is not verifiable, then an action is chosen to establish the verifiability: possible actions are either a decomposition of $\rho\mathrm{M}_1$ in order to obtain a verifiable submethod or the calculation of additional preconditions of the method yielding verifiability.

---

input: linearised source plan, (open) target goal
output: (linearised) target plan

1. **while** there are open target goals **do**
2. **if** source plan is exhausted, **then** base-level plan for the open goals, **else**
3. Get next sequent P from source plan. The sequent is either an assumption or a goal. **if** P is an assumption, **then** go to 5.
4. **if** there is a reformulation ρ, such that $\rho\mathsf{P}$ matches an open target goal g_T for which justifications hold, **then**
   - reformulate source plan by ρ and link g_T to source plan.
   - **if** g_T is an open goal, **then**
     - Select from the reformulated source relevant b-method M.
     - **if** M's justifications hold, **then** transfer M to the target plan and update open goals.
     - **if** justifications do not hold, **then** choose suitable **action**:
       - Try to establish justifications by other means.
       - Or base-level plan.
5. Select from the reformulated source the relevant f-method M.
6. **if** there is a reformulation ρ' left such that $|missing(\rho'\mathrm{M})| < m$ and such that justifications hold for the matched target assumptions **then**
   - reformulate source plan by ρ' and link the matched target assumptions to source plan.
   - **if** ρ'M's justifications hold, then transfer the method to the target and update open goals and assumptions.
   - **if** justifications do not hold, **then** choose suitable action as above.

Table 1: Outline of the analogy-driven proof-plan construction

The analogy procedure is repeated, first testing termination conditions (1.,2.). Base-level planning is activated when the guidance by the source proof-plan is exhausted in order to prove the remaining open goals. Source methods that become redundant in the target are skipped. This procedure yields a target plan with verified methods. The target plan may have open goals which eventually have to be closed by base-level planning.

[4] For *missing(M)* := set of preconditions of M that do not match a current target assumption.

3 Example

Woody Bledsoe [Bledsoe, 1994] provided the Heine-Borel example as a challenge problem for theorem proving by analogy that could not be solved by previous approaches.

THEOREM: Heine-Borel-1 (HB1) *If a closed interval* $[a, b]$ *of* R^1 *is covered by a family* G *of open sets (in* R^1*), then there is a finite subfamily* H *of* G *which covers* $[a, b]$.[5] ■

Formalised:
$\{a \in R \wedge b \in R \wedge a \leq b, \forall B(B \in G \rightarrow open(B)), [a, b] \subset \cup G\}$
$\vdash \exists H(H \subseteq G \wedge finite(H) \wedge [a, b] \subset \cup H)$

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

Formalised:
$\{a \in R \wedge b \in R \wedge c \in R \wedge d \in R \wedge a \leq b \wedge c \leq d, \forall B(B \in G \rightarrow open(B)), [a, b, c, d] \subset \cup G\}$
$\vdash \exists H(H \subseteq G \wedge finite(H) \wedge [a, b, c, d] \subset \cup H)$

To give an idea, a (nontrivial) mathematical proof of HB1 by contradiction goes like this (using lemmas listed below):
A sequence $g(i)$ is defined by
$g(0) = [a, b]$

$$g(i+1) = \begin{cases} [lf(g(i)), mid(g(i))] & : (*) \\ [mid(g(i)), rt(g(i))] & : \text{otherwise} \end{cases}$$

where (*)= if no finite subset of G covers $[lf(g(i)), mid(g(i))]$ and
$mid(g(i)) = lf(g(i)) + (lf(g(i)) - rt(g(i)))/2$[6]. By induction on i (which we deliberately circumvent in our proof-plan) it is shown that $g(i)$ is a nested sequence of closed intervals and that for each $i \in \mathbb{N}$ no finite subset of G covers $g(i)$ and $(rt(g(i)) - lf(g(i))) = (b - a)/2^i$. Since $g(i)$ is a nested sequence of non-empty closed intervals, by the Nested Interval Lemma (Nit1) there is a z for which $z \in g(i)$ for each $i \in \mathbb{N}$. Thus, $z \in [a, b]$ and since $[a, b] \subset \cup G$, it follows by Lemma6 that $z \in B$ for some $B \in G$. Since B is open, Lemma7 yields that there is a closed interval $[u, v]$ for which $u < v, [u, v] \subset B$, and $z = mid([u, v])$. By Lemma8 choose i such that $(b - a)/2^i < (u - v)/2$. By Lemma9 follows then that $g(i) \subset [u, v] \subset B$ and thus $g(i) \subset \cup(\{B\})$ by Lemma12, where $\{B\}$ is a finite subset of G.

This example has been proceeded in detail in [Melis, 1995a]. Here we cannot go into detail and just explain the bottom line. Figure 1 shows the proof-plan for HB1 which, in fact, yields a proof of HB1 when executed. One of its methods, method-111, is shown below.

3.1 Proving HB2 by Analogy

L1, L2, L6, L8, L10, L12, L13, L14, L17 in Figure 1 are potential lemmas for HB2 because they contain only symbols not specific for R^1. The parameter m of the

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

[6] lf, rt, and mid denoting the lower, upper bound and the middle of an interval

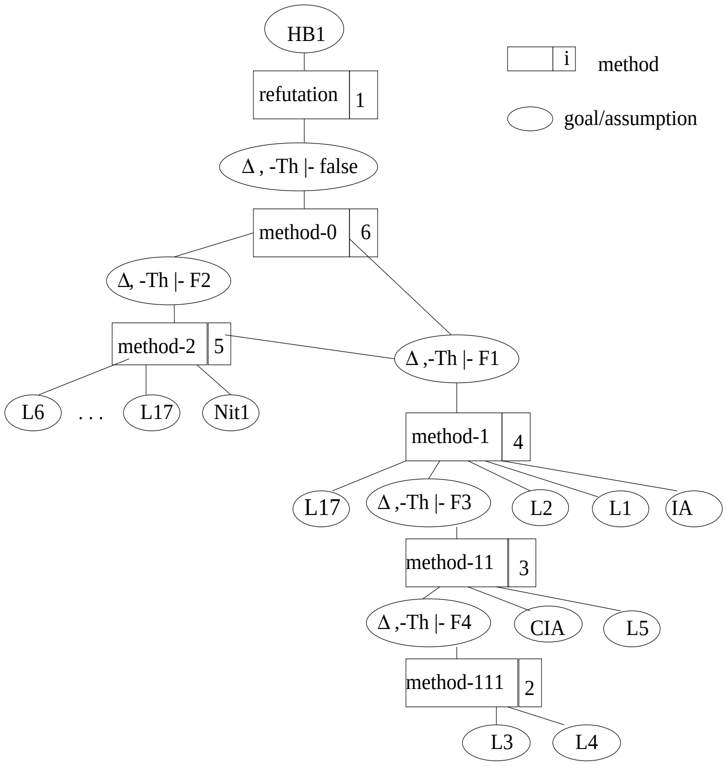

Figure 1: The proof-plan of HB1 with annotated sequence of method nodes

analogy procedure is set to a large number because many lemmas are missing for HB2.

The reformulations are assisted by a connection-table `CT`, which contains connections of the kinds, $([-,-][-,-,-,-])$, $(clsdint\ clsdrect)$, and $(R^1 R^2)$, where $[-,-]$ is a function that maps two real numbers x, y to the real interval $[x, y]$ and $[-,-,-,-]$ maps four real numbers to the two-dimensional interval $[x, y, w, z]$ [7]. `CT` restricts the search for reformulations and reduces the number of parameters that have to be instantiated to be able to prove suggested target lemmas after the analogy procedure. `CT` is a *permanent* connection table, which contains information for a large area of mathematics, and can be used for other analogy theorem proving problems as well.

The source proof-plan is reformulated stepwise along with a step by step transfer of methods to the intermediate plan.

1. The source goal HB1 has to be reformulated such that it matches HB2. The reformulation consists of the `Symbol-Mapping` instantiating the parameters $a, b, G, clsdint, open, finite$ to the constants $a, b, G, open, finite$ of HB2 and of `Add-Arguments`. The latter reformulation changes the binary function $[-,-]$ to the 4-ary function $[-,-,-,-]$ (see, e.g., [Melis, 1995c]). The indirect proof method and some subgoals and assumptions stay unchanged but all other methods are changed by the reformulation:

[7] $\lambda x \lambda y.[x, y]$ and $\lambda x \lambda y \lambda w \lambda z.[x, y, w, z]$ respectively.

For instance, it changes lines in proof schemas such as
$\Delta \vdash a \in R \wedge b \in R \rightarrow clsdint([a,b])$ to
$\Delta' \vdash a \in R \wedge b \in R \wedge c \in R \wedge d \in R \rightarrow clsdint([a,b,c,d])$
and an assumption
$clsdint([a,b]) \wedge \neg\exists H(H \subseteq G \wedge finite(H) \wedge [a,b] \subset \cup H) \wedge lf([a,b]) \leq rt([a,b])$ to
$clsdint([a,b,c,d]) \wedge \neg\exists H(H \subseteq G \wedge finite(H) \wedge [a,b,c,d] \subset \cup H) \wedge lf([a,b]) \leq rt([a,b]) \wedge lf'([a,b,c,d]) \leq rt'([a,b,c,d])$.[8] The indirect proof method is verifiable and, thus, is transferred to the target.

2. Next, the source assumption L3 (reformulated in step 1) leads the f-method method-111'. No lemmas corresponding to L3, L4 are given for the target, since the parameter m of the analogy procedure is large, this does not matter. $clsdint, lf, rt, lf', rt'$ occur in the reformulated L3,L4, thus the `Symbol-Mapping` $clsdint \Rightarrow clsdrec$ is forced by `CT`. The parameter lf, rt, lf', rt' cannot be instantiated and remain parameters. method-111 is reformulated to method-111' by the reformulations of step 1 and 2 (See below.). method-111' is verifiable and is transferred to the target plan.

3. The next (previously reformulated) source sequent is $(\Delta', \neg Th' \vdash F4')$ and this sequent is related the f-method method-11' in the reformulated source plan. No lemmas corresponding to the (reformulated) source assumptions L5', CIA' are given in the target which does not matter because of the large parameter m. No reformulations are necessary. method-11', shown below, is verifiable and is transferred to the target plan.

4. Essentially the same is true for the next assumption and method-1'.

5. Now it is method-2's turn. $center$ occurs in the preconditions of method-2' and remains a parameter since no lemma to match is given in the target. No reformulation takes place.
   - Checking the justifications of method-2' we find that method-2' is not verifiable because line2'-5
     $\Delta', \neg Th' \vdash$
     $\exists i(i \in \mathbb{N} \wedge (rt([a,b,c,d]) - lf([a,b,c,d]))/2^i \leq (rt([u,v,s,t]) - lf([u,v,s,t]))/2 \wedge (rt'([a,b,c,d]) - lf'([a,b,c,d]))/2^i \leq (rt'([u,v,s,t]) - lf'([u,v,s,t]))/2)(\ldots)$ is not. In order to establish the verifiability method-2' is decomposed into a plan consisting of the verifiable method-21' with post(method-2') = post(method-21'), the subgoal $\mathtt{g'}_{5a}$ which is in pre(method-2'), and a not verifiable method-22' with post(method-22') = $\mathtt{g'}_{5a}$.
   - method-21' can be transferred and $\mathtt{g'}_{5a}$ remains an open goal.

6. After method-21' is transferred to the target plan, $(\Delta', \neg Th' \vdash F2)$ is the next source assumption and leads to method-0'. No further reformulation happens in this step. method-0' is verifiable and can be transferred to the target plan.

[8] `Add-Arguments` does not introduce introduce additional subplans in the Heine-Borel example.

3.2 The Resulting HB2 Proof Plan

The analogy procedure yields the copy of the source plan, shown in Figure 2, with reformulated goals, assumptions and verifiable methods. The target plan for HB2 looks exactly like that for HB1 except that some new lemmas $L3' \ldots$ replace $L3 \ldots$ and all methods but method-2 are replaced by the corresponding reformulated methods. method-2 is replaced by its reformulated submethod method-21'. Actually this proof-plan has been produced by interactively choosing the reformulations but applying it automatically. The analogy procedure suggests open goals (lemmas) L3', L4', L5', L7', L9', L11', Nit2, CIA', IA', g_{5a} which are left to be proved.

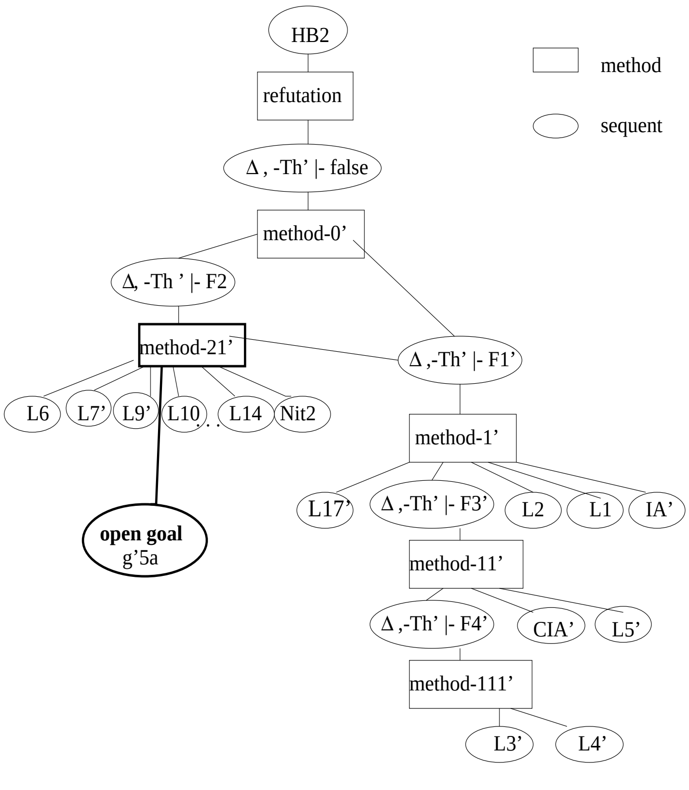

Figure 2: The proof-plan of HB2

Usually one has to find an appropriate interpretation of the parameters not instantiated yet, i.e., the symbols not occurring in the target theorem or target assumptions, nor in `CT` in order to be able to accomplish the proofs of the open goals. In the Heine-Borel example an appropriate interpretation of lf by lf, of rt by rt, of lf' by bot, of rt' by top, of $center$ by $center$ in R^2 makes the open goals true. Assuming the open goals as proof assumptions of HB2, the proof-plan actually yields a proof of HB2 when executed.

method: `-111`	
parameter	$a, b, G, clsdint, open, rt, lf, [-,-], finite$
preconditions	L3,L4
postcondition	$(\Delta, \neg Th \vdash F4)$
constraints	
proof schema	0-1. $\Delta \vdash a \in R \wedge b \in R \wedge a \leq b$ (HYP) 1. $\Delta \vdash a \in R \wedge b \in R$ ($\wedge$E,0-1) 2. $\Delta \vdash a \in R \wedge b \in R \rightarrow clsdint([a,b])$ ($\forall$E;L3) 3. $\Delta \vdash clsdint([a,b])$ ($\rightarrow$E,1,2) 4. $\{\neg Th\} \vdash \neg \exists H(H \subseteq G \wedge finite(H) \wedge [a,b] \subset \cup H)$ (HYP) 5. $\Delta \vdash lf([a,b]) \leq rt([a,b])$ (OTTER;L4, 1) 6. $\Delta \vdash [a,b] \subset \cup G$ (HYP) 7. $\Delta, \neg Th \vdash F4$ ($\wedge$I,3,4,5,6)
procedure	schema-interpreter

method: `-111'`	
parameter	rt, lf, rt', lf'
preconditions	L3',L4'
postcondition	$(\Delta', \neg Th' \vdash F4')$
constraints	type(rt)=type(rt'), type(lf)=type(lf')
proof schema	0-1. $\Delta' \vdash a \in R \wedge b \in R \wedge a \leq b \wedge c \in R \wedge d \in R \wedge c \leq d$ (HYP) 1. $\Delta' \vdash a \in R \wedge b \in R \wedge c \in R \wedge d \in R$ ($\wedge$E,0-1) 2. $\Delta' \vdash a \in R \wedge b \in R \wedge c \in R \wedge d \in R \rightarrow clsdrec([a,b,c,d])$ ($\forall$E;L3') 3. $\Delta' \vdash clsdrec([a,b,c,d])$ ($\rightarrow$E,1,2) 4. $\{\neg Th'\} \vdash \neg \exists H(H \subseteq G \wedge finite(H) \wedge [a,b,c,d] \subset \cup H)$ (HYP) 5. $\Delta' \vdash lf([a,b,c,d]) \leq rt([a,b,c,d]) \wedge lf'([a,b,c,d]) \leq rt'([a,b,c,d])$ (OTTER;L4', 1) 6. $\Delta' \vdash [a,b,c,d] \subset \cup G$ (HYP) 7. $\Delta', \neg Th' \vdash F4'$ ($\wedge$I,3,4,5,6)
procedure	schema-interpreter

method: `-11`	
parameter	$a, b, G, clsdint, open, rt, lf, [-,-], finite, g$
preconditions	$(\Delta, \neg Th \vdash F4)$,L5,CIA
postcondition	$(\Delta, \neg Th \vdash F3)$
constraints	
proof schema	1. $\Delta, \neg Th \vdash F4$ (LEMMA) 2. $\Delta, \neg Th \vdash \exists g(g(0) = [a,b] \wedge \forall i(i \in \mathbb{N} \rightarrow P(g(i)) \wedge Q(g(i), g(i+1))))$ (OTTER;L5, CIA,1) 3. $\Delta, \neg Th \vdash F3$ ($\exists$E;2)
procedure	schema-interpreter

method: `-11'`	
parameter	rt, lf, rt', lf'
preconditions	$(\Delta', \neg Th' \vdash F4')$,L5',CIA'
postcondition	$(\Delta', \neg Th' \vdash F3')$
constraints	type(rt)=type(rt'), type(lf)=type(lf')
proof schema	1. $\Delta', \neg Th' \vdash F4'$ (LEMMA) 2. $\Delta', \neg Th' \vdash \exists g(g(0) = [a,b,c,d] \wedge \forall i(i \in \mathbb{N} \rightarrow P'(g(i)) \wedge Q'(g(i), g(i+1))))$ (OTTER;L5', CIA',1) 3. $\Delta', \neg Th' \vdash F3'$ ($\exists$E;2)
procedure	schema-interpreter

4 Discussion

The robustness of the analogy-driven proof-plan construction to different formulations of theorems and possible improvements have to be investigated. Actually, we looked at three different formulations when we were dealing with the Heine-Borel examples[9]:

1. Let the two theorems Heine-Borel-1 and Heine-Borel-2 be formalised by HB1:
$\ldots \vdash clsdint(I) \rightarrow \exists H(H \subseteq G \wedge finite(H) \wedge D \subset \cup H)$
and HB2:
$\ldots \vdash clsdrec(I) \rightarrow \exists H(H \subseteq G \wedge finite(H) \wedge D \subset \cup H)$,
and the some source assumption
$\emptyset \vdash \forall D(clsdint(D) \rightarrow \exists x, y(D = [x, y] \wedge x \in R \wedge y \in R))$
and the target assumption
$\emptyset \vdash \forall D(clsdrec(D) \rightarrow \exists x, y, z, w(D = [x, y, z, w] \wedge x \in R \wedge y \in R)) \wedge z \in R \wedge w \in R$.
When the function $[-, -]$ occurs first in the proof-plan, the connection table forces its mapping to $[-, -, -, -]$ and this triggers the reformulation `Add-Arguments` only later than in the first analogy step. The first occurrence of the function $[-, -]$ depends, however, on the actual proof of HB1.

2. If HB1 is formalised as
$\ldots clsdint(I) \vdash \exists H(H \subseteq G \wedge finite(H) \wedge D \subset \cup H)$ and HB2 as
$\ldots \vdash \exists H(H \subseteq G \wedge finite(H) \wedge [a, b, c, d] \subset \cup H)$,
then a normalising reformulation (not discussed in this paper), has to be applied to the source plan in order to match HB1 with HB2. Such a normalisation introduces an additional method at the top of the source proof-plan that applies the definition of $clsdint(X)$ and yields a postcondition that can be matched with HB2.

3. We ran into problems when we tried to use $length(D)$ instead of $rt(D) - lf(D)$ in the assumptions, because, for instance, a target lemma
L9'= $\emptyset \vdash \forall x \forall u \forall v \forall s \forall t \forall c \forall d \forall e \forall f(u \in R \wedge v \in R \wedge s \in R \wedge t \in R \wedge c \in R \wedge d \in R \wedge e \in R \wedge f \in R \wedge u \leq v \wedge s \leq t \wedge c \leq d \wedge e \leq f \rightarrow (center(x, [u, v, s, t]) \wedge x \in [c, d, e, f] \wedge length([c, d, e, f]) \leq length([u, v, s, t])/2 \wedge length([c, d, e, f]) \leq length([u, v, s, t])/2 \rightarrow [c, d, e, f] \subset [u, v, s, t]))$
was suggested for which we did not know how to interpret *length* in order to satisfy the lemma.

More work is necessary to find powerful and general reformulations that can depend on the actual context, particularly normalising and abstracting reformulations. Presumably, meta-planning with meta-methods is necessary to automate the choice of sequences of reformulations. The HB1-methods have been user supplied and encode partial proof strategies themselves. Taking into consideration that the actual OTTER procedure can, of course, not be reformulated, the resemblance to plans from other planners, such as *CIAM* [van Harmelen *et al.*, 1993], becomes more obvious. That means, our concept of reformulations is not bound to the presented type of methods.

[9]These examples were not proceeded in detail though.

The approach seems to be general enough to be applicable in another proof planner *CIAM*. Ongoing experiments with *CIAM* for efficiency reasons use matchings that are be restricted by domain-specific heuristics (i.e., for induction) and they have shown that many but not all reformulations are dependent on the domain and its heuristics. Since usually little search for methods is necessary in *CIAM* because of the encoded strong heuristics of inductive theorem proving, the main advantage of analogy cannot be to save search for methods in *CIAM*. However, the justifications are elaborate for *CIAM*'s methods and the actions to establish justifications play a major role for the usefulness of analogy in this framework.

5 Conclusion

We surely did not solve all problems that occur in proving theorems by analogy but we made considerable progress because the new approach succeeded with analogy problems that previous approaches were not able to cope with. The success of proving the Heine-Borel-2 theorem by analogy is mainly due to the application of reformulations.

Previous approaches to analogy in theorem proving basically used just symbol mapping and did not attempt to re-represent, abstract, or elaborately reformulate the source problem or the target problem and, hence, their results were highly dependent on the actual representation of theorems and proof assumptions. In contrast, our analogy employs reformulations that go beyond symbol mapping and benefits of certain normalising and abstracting reformulations that re-represent and abstract the source problem and change the source proof-plan accordingly.

The need to incorporate re-representations and abstraction into analogy was also pointed out, e.g., by Indurkhya [Indurkhya, 1992; Indurkhya and O'Hara, 1993] and Plaisted [Plaisted, 1981; Giunchiglia and Walsh, 1992; Villafiorita and Sebastiani, 1994], respectively. Since re-representations and useful abstractions are at least domain-dependent it is a problem to incorporate these general ideas into an analogy system. In order to avoid this problem, abstractions are user-supplied in [Villafiorita and Sebastiani, 1994] or frequently needed normalisations and abstractions are predefined as in our model.

The reformulation and transfer at the proof-plan level turned out to be better suited for analogy in theorem proving than that at the level of calculus proofs.

6 Acknowledgement

Special thanks to Woody Bledsoe for helping with the Heine-Borel example. He first suggested that we collaborate on this example and recommended the HB1, HB2 pair, proposed a proof-plan of HB1 with verified methods and a lemma list of HB1 [Bledsoe, 1994], and implemented the reformulation `Add-Arguments` according to the given description. Actually he should have been a coauthor but he decided not to coauthor because of health problems.

References

[Bledsoe, 1986] W.W. Bledsoe. The use of analogy in automatic proof discovery. Tech.Rep. AI-158-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.

[Bledsoe, 1994] W.W. Bledsoe. Heine-Borel theorem analogy example. Technical Report Memo ATP 124, University of Texas Computer Science Dept, Austin, TX, 1994.

[Bledsoe, 1995] W.W. Bledsoe. A precondition prover for analogy. *BioSystems*, 34:225–247, 1995.

[Brook *et al.*, 1988] 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.

[Bundy, 1988] A. Bundy. The use of explicit plans 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.

[Carbonell, 1986] 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.

[Giunchiglia and Walsh, 1992] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–390, 1992.

[Huang *et al.*, 1994a] X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Omega-MKRP: A Proof Development Environment. In *Proc. 12th International Conference on Automated Deduction (CADE)*, Nancy, 1994.

[Huang *et al.*, 1994b] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Methods - the basic units for planning and verifying proofs. In *Proceedings of Jahrestagung für Künstliche Intelligenz*, Saarbrücken, 1994. Springer.

[Indurkhya and O'Hara, 1993] Bipin Indurkhya and Scott O'Hara. Incorporating (re)-interpretation in case-based reasoning. In M.M. Richter, S. Wess, K-D. Althoff, and F. Maurer, editors, *EWCBR-93 First European Workshop on Case-Based Reasoning*, Kaiserslautern, 1993.

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

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

[Kolbe and Walther, 1995] Th. Kolbe and Ch. Walther. Patching proofs for reuse. In N. Lavrac and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning 1995*, Kreta, 1995.

[McCune, 1990] W.W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.

[Melis, 1995a] E. Melis. Analogy-driven proof-plan construction. Technical Report DAI Research Paper No 735, University of Edinburgh, AI Dept, Dept. of Artificial Intelligence, Edinburgh, 1995.

[Melis, 1995b] E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

[Melis, 1995c] E. Melis. Modification in analogy-driven proof-plan construction. In *Proceedings of the IJCAI-95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, Montreal, 1995.

[Mostow, 1989] J. Mostow. Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40(1-3):119–184, 1989.

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

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

[Plaisted, 1981] D. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[Polya, 1957] G. Polya. *How to Solve it*. 2nd ed. Doubleday, New York, 1957.

[van Harmelen *et al.*, 1993] F. van Harmelen, A. Ireland, S.Negrete, A. Stevens, and A. Smail. The CLAM proof planner, user manual and programmers manual. Technical Report version 2.0, University of Edinburgh, Edinburgh, 1993.

[Veloso, 1994] M.M. Veloso. Flexible strategy learning: Analogical replay of problem solving episodes. In *Proc. of the twelfth National Conference on Artificial Intelligence 1994*, Seattle, WA, 1994.

[Villafiorita and Sebastiani, 1994] A. Villafiorita and R. Sebastiani. Proof planning by abstraction. In *Proceedings ECAI-94 workshop on interactive systems*, Amsterdam, 1994.

[Wos, 1988] L. Wos. *Automated Reasoning: 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, 1988.