Theorem Proving by Analogy – A Compelling Example*

Erica Melis

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

Abstract. This paper shows how a new approach to theorem proving by analogy is applicable to *real maths* problems. This approach works at the level of proof-plans and employs reformulation that goes beyond symbol mapping. The Heine-Borel theorem is a widely known result in mathematics. It is usually stated in R^1 and similar versions are also true in R^2, in topology, and metric spaces. Its analogical transfer was proposed as a challenge example and could not be solved by previous approaches to theorem proving by analogy. We use a proof-plan of the Heine-Borel theorem in R^1 as a guide in automatically producing a proof-plan of the Heine-Borel theorem in R^2 by analogy-driven proof-plan construction.

1 Introduction

Theorem proving by analogy has been clearly recognized as a powerful heuristic in mathematical problem solving [16, 18]. In automated and interactive theorem proving, analogy is particularly useful in situations where much search is necessary, e.g., where many proof assumptions or long proof paths are involved. Analogy in automated theorem proving is, however, still a challenging problem [1, 20].

Previous approaches to theorem proving by analogy had problems with real, complex maths examples because they have been dominated by the idea of mapping symbols of the source theorem to symbols of the target theorem and employing an extended symbol map for transferring single calculus level proof steps of the source proof[3]. Empirical investigations [12], however, have provided evidence that this idea does not appropriately cover the analogies drawn by mathematicians, not even all the analogies in the textbook [5].

In [14] we developed a new approach to theorem proving by analogy and explained why the *plan* level is an appropriate one for the analogical transfer. As shown in [11] this method can cope with the analogies in the mentioned textbook. In order to prove this approach to be successful, we now show that, unlike

* paper at EPIA-95. This work was supported in part by the HC&M grant CHBICT930806 and by a research grant of the Deutsche Forschungsgemeinschaft.

** On leave from University Saarbrücken, Germany

[3] Though Bledsoe transfers larger steps calling an automated prover but again uses symbol mapping.

other approaches, it is even able to deal with more complicated maths theorems that are considered challenge. Our method can handle a non-trivial Heine-Borel theorem, proposed in [2] as a challenging example for theorem proving by analogy. This example could not be solved by previous approaches to analogy in theorem proving.

The paper is organized as follows: First the new approach is presented. Then we introduce the challenge example and describe its analogy-driven proof-plan construction step by step.

2 The Approach

Our method transfers proof-plans analogically and incorporates the reformulation of problems and operators that may include abstraction and other reformulations different from symbol mapping. For employing the full range of reformulations, the operator representation has to be mainly declarative. Therefore we describe the operators designed for Ω-MKRP [6]. We briefly review how the specific operators and plans are defined, introduce reformulation, and discuss the analogy procedure.

2.1 Preliminaries

Proof planning operators have been introduced by Bundy [3]. Our planning operators, called *methods*, are frame-like structures defined in [7] with pre- and postconditions just as the common planning operators. More specifically, methods M have the following slots: parameter, preconditions (pre(M)), postcondition (post(M)), constraints, proof schema and procedure. pre(M) is a set of sequents[4] from which the application of the method derives post(M) which is a set of sequents 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 standard program in the slot procedure executes the application of the proof schema. The proof schema is a declarative schematic representation of proofs in the object logic, relying on the Natural Deduction (ND) calculus and on invoking automated theorem provers such as OTTER [10]. The proof schema lines contain a label, a sequent, and a line-justification. The line-justification consists of the name of an ND-rule, the name of a prover, or LEMMA in case, the sequent is in pre(M). Additionally it may include supporting lines. For instance,

3. $\Delta \vdash F$ (IP;2)

describes that $\Delta \vdash F$ is derived from the sequent in line 2 of the proof schema by the ND-rule IP[5]. An example of a method is

[4] *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.

[5] The IP rule is a combination of the ND-rules $\neg$I and $\neg$E that has been proved to be correct.

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

Our methods mainly differ from those in [3] in that the tactic slot is replaced by a declarative proof schema *and* a procedure interpreting this schema[6]. The intention behind this difference is to enable reformulations of methods.

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

Since 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.

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 $\mathtt{g}$ and precedes the sequent nodes $\mathtt{g}_1, \ldots, \mathtt{g}_n$ if
$\mathtt{g} \in \sigma(\text{post(M)})$ and $\sigma(\text{pre(M)}) = \{\mathtt{g}_1, \ldots, \mathtt{g}_n\}$ for a substitution σ of parameters.

Proof planning starts with a goal $\mathtt{g}$ and assumptions $(\emptyset \vdash F_i)$, where F_i are proof-assumptions, axioms, definitions, or lemmata. 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 $\mathtt{g}_i$ equals an assumption, the two nodes collapse. Then $\mathtt{g}_i$ 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 *linearized* proof-plans ordered by the sequence in which the nodes have been

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

added to the plan. As in [19] justification structures, used to encode justifications for the decision made, annotate the plan nodes. These *justifications* capture the subgoaling structure of a plan and point to reasons for the choice, such as application conditions of a method, user-given guidance, or pre-programmed control knowledge. The verifiability of a method is one of the justifications.

Reformulations are mappings ρ of a proof-plan to a proof-plan which usually but not necessarily preserve the verifiability of methods in the plan. They encode mathematical heuristics on how a proof-plan changes dependent on certain changes of the problem to be proved. Reformulations may insert subplans or replace methods and may change methods, sequents, and justifications of plan nodes. 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.

The reformulation `Add-Arguments` is applied in the example below. It is applicable if a function f is to be mapped to a function f', where arguments x_i (or tuples of arguments) of f are mapped to k_i arguments (or tuples of arguments) x_{ij} of f'; for instance, if the function $[-,-]$ mapping a pair of real numbers x, y to a real interval $[x, y]$ is mapped to a function $[-,-,-,-]$ with two pairs $(x, y), (z, w)$ of real numbers as arguments that yields an interval $[x, y, z, w]$ in R^2. The program of `Add-Arguments` replaces the function f by f' with duplicated arguments in goals, assumptions, justifications, and methods. It yields additional related changes in the proof schema of effected methods such as replacing certain subformulas by conjunctions and duplicating certain lines of the proof schema which may also cause the duplication of related preconditions of the method (see [13] for more details). If a method's precondition P became duplicated, then `Add-Arguments` modifies the structure of the proof-plan by duplicating the subplan that yields the goal/assumption P in the original proof-plan.

For example, for the functions $[-,-]$ and $[-,-,-,-]$ to be mapped and $(x, y), (z, w)$ corresponding to (x, y) a proof-schema line
$\emptyset \vdash a \in R \wedge b \in R \rightarrow clsdint([a, b])$ would be changed to the line
$\emptyset \vdash a \in R \wedge b \in R \wedge c \in R \wedge d \in R \rightarrow clsdint([a, b, c, d])$ and

$\emptyset \vdash \forall x \forall y \forall z \forall w(lf([x, y, z, w]) = x \wedge rt([x, y, z, w]) = y)$ is replaced by
$\emptyset \vdash \forall x \forall y \forall z \forall w(lf([x, y, z, w]) = x \wedge lf'([x, y, z, w]) = z \wedge rt([x, y, z, w]) = y \wedge$
$rt'([x, y, z, w]) = w)$

Even so it might seem that `Add-Arguments` was designed exactly for the Heine-Borel example or that this particular example was chosen according to the reformulation, this is *not* the case. `Add-Arguments` proved to be a fairly frequent and general reformulation that has been used in several examples and was documented in [11, 7]. Moreover, the particular Heine-Borel example was chosen independently in [2].

2.2 Analogy-Driven Proof-Plan Construction

The general idea of analogy-driven proof-plan construction is to use the linearized source proof-plan as a guide for constructing an analogous target proof-plan, to reformulate the source plan, and to transfer methods, goals, and assumptions of a reformulated source proof-plan to the target proof-plan. The analogy-driven proof-plan construction is a derivational analogy [4, 19], which has not been used for theorem proving before, extended by reformulation and bidirectional planning.

During the analogical transfer it is checked whether the justifications from the source node hold for the corresponding node in the target, e.g. whether the application conditions hold in the target. Checking the justifications makes it possible to consider information that is not available in the source and target problems, but relevant to the proof. This idea goes back to Carbonell's derivational analogy [4], where a decision in the target is made correspondingly to the decision in the source only if the justifications of the decision hold in the target as well. Thereby the requirement of a semantic justification of analogical reasoning [17] can be met.

Our analogy employs reformulation that aims at matching a source goal with a target goal or as many preconditions of a source f-method with target assumptions respectively and that yield corresponding changes of the proof-plan. Table 1 shows the top-level procedure of the analogy-driven proof-plan construction. Given a linearized 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 whereas step 5 and 6 apply to the treatment of forward planning. The two branches differ mainly in that ρ aims at matching a source goal to one target goal, whereas ρ' aims at matching as many source assumptions as possible to target assumptions. $|missing(\rho' \mathrm{M})| < m$[7] means that less than m preconditions of the currently treated reformulated f-method M do not match a target assumption. The sequents of $missing(\rho' \mathrm{M})$ become new open goals if ρ' is an acceptable reformulation. m is a procedure parameter and indicates the confidence in an analogical transfer despite mising target lemmata.

According to the order of the source plan the first goal, usually the source problem P_S, is chosen. 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 and the method M with post(M) $= \rho\mathsf{P}_S$ is a candidate for the transfer to the target plan. If the original justifications hold for the M in the target, then M is transferred and the open goals are updated by removing post(M) and introducing all sequents from pre(M) as open goals. The procedure is repeated, first testing termination reasons.

If the justifications do not hold for M in the target, then different actions can be taken dependent on the kind of violated justification. For instance, a

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

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

1. **while** there are open target goals **do**
2. **if** source plan is exhausted, **then** base-level plan for the open goals.
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 ρP matches an open target goal g_T for which the justifications hold, **then**
   - reformulate source plan by ρ and link g_T to source plan.
   - Select from the reformulated source the relevant b-method M.
   - Check M's justifications.
   - **if** justification does not hold, **then** choose suitable action:
     - Advance the source if M is not necessary.
     - Try to establish the 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 that justifications hold for the matched target assumptions **then**
   - reformulate source plan by (best) ρ' and link the matched target assumptions to source plan.
   - Check M's justifications.
   - **if** justification does not hold, **then** choose suitable action as above.

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

reformulated method might not be verifiable because reformulations do not necessarily preserve verifiability. If the verifiability-justification does not hold for M, then try to *modify* M to a verifiable method M'. One possible modification technique decomposes M such that a verifiable submethod M' results with post(M)= post(M') and M' replaces M. This technique creates a new open goal as well. If a precondition of M does not hold in the target, then it can be made an open goal that has to be proved by other means (e.g., base-level planning).

Another action is taken if the current goal is satisfied in the target already. Then M is superfluous in the target and can be skipped. The last option is to replace M by another method M', and this requires base-level planning for M'.

When all methods of the source plan have been visited, then the remaining open goals have to be proved (by base-level planning). This situation is similar to human theorem proving by analogy that usually leaves new details to prove.

The argument that every new example would need a new reformulation did not prove to be true during experiments with the analogy procedure. In fact, few normalizing and abstracting reformulations were used and most often `Symbol-Mapping`, `Term-Mapping`, and `Add-Arguments` were used (see [11, 15]). This claim needs, however, further experience and evaluation.

3 The Heine-Borel Example

Theorem 1 Heine-Borel-1 (HB1). *If a closed interval $[a,b]$ of R^1 is covered by a family G of open sets (in R^1), then there is a finite subfamily H of G which covers $[a,b]$.*[8]

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

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

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

Figure 1 shows the complex proof-plan for HB1 which, in fact, yields a proof of HB1 when executed.

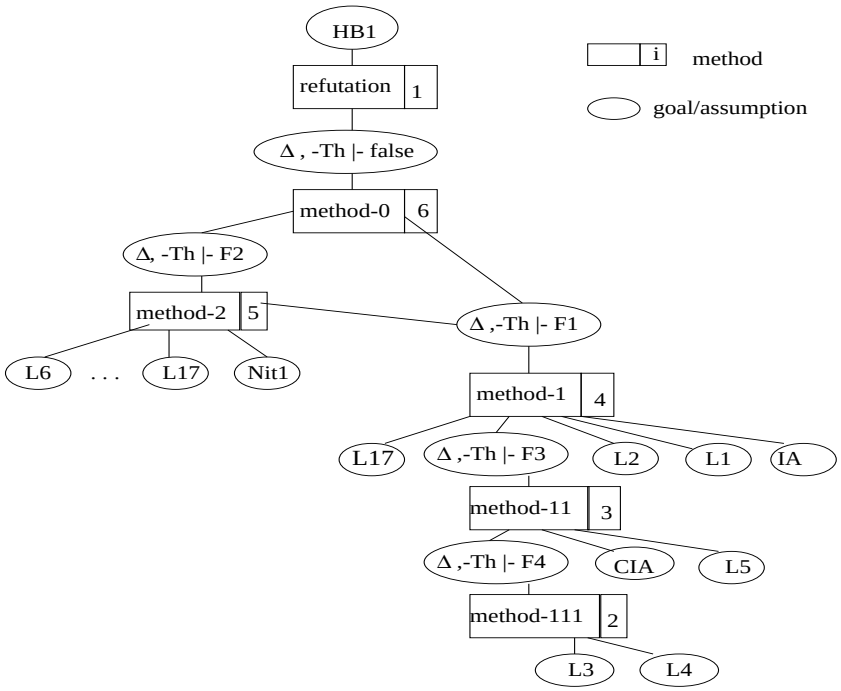

Fig. 1. The proof-plan of HB1 with annotated sequence of method nodes

[8] 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.

One of its (verifiable) methods, method-111, is shown below. This example has been proceeded detail in [13]. Here we cannot go into detail and just explain the bottom line.

3.1 Proving HB2 by Analogy

The lemmata of HB1 L1, L2, L6, L8, L10, L12, L13, L14, L17 are potential lemmata for HB2 as well because they contain only symbols not specific for R^1. m is set to a large number because many lemmata are missing for HB2.

The reformulations are assisted by a user-supplied connection-table `CT`, which contains connections of the kinds, ([-,-] [-,-,-,-]), (*clsdint clsdrect*), and $(R^1 R^2)$. (Here [-,-] is written for $\lambda x, y.[x, y]$ and [-,-,-,-] for $\lambda x, y \lambda z, w[x, y, z, w]$). In our example the association $[-,-][-,-,-,-]$ also points to an association of the pair (x, y) with the two pairs $(x, y)(z, w)$.

`CT` restricts the search for reformulations and reduces the number of parameters that have to be instantiated to be able to prove suggested target lemmata after the analogy procedure. `CT` is a *permanent* connection table, which contains information for an area of mathematics, and can be used for other analogy problems as well. Not surprisingly, `CT` may carry *semantic* information which often is important for the support of analogical transfer in mathematics. In fact, the association of (x, y) with $(x, y), (z, w)$ is a semantic information.

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

1. The source goal HB1 has to be reformulated such that it matches HB2. The reformulation consists of the `Symbol-Mapping` instantiating the parameters $a, b, G, open, finite$ to the constants $a, b, G, open, finite$ of HB2 and of `Add-Argument` which inter alia replaces the binary function $[-,-]$ by the 4-ary function $[-,-,-,-]$ and introduces the additional parameters lf', rt' and constants c, d. No duplication of subplans occurs in this application of `Add-Argument`. The indirect proof method stays unchanged but all other methods are changed by the reformulation.
2. Next, the source assumption L3 (reformulated in step 1) leads the f-method method-111'. No lemmata corresponding to L3, L4 are given for HB2 but, since m is large, this does not matter. The parameter $clsdint, lf, rt, lf', rt'$ occur in the reformulated L3, L4, thus the `Symbol-Mapping` $clsdint \Rightarrow clsdrec$ is forced by `CT` while lf, rt, lf', rt' cannot be instantiated and remain parameters. method-111 is reformulated to the verifiable method-111' by the reformulations of step 1 and 2:

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

3. The next (previously reformulated) source sequent is $(\Delta', \neg Th' \vdash F4')$ and the corresponding f-method is method-11'. No lemmata corresponding to L5',CIA' are given in the target which does not matter because of the large m. No reformulations are necessary. method-11' is verifiable.
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 L7 or L9 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 justification 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. $(\Delta', \neg Th' \vdash F2)$ is the next source assumption and leads to method-0'. No further reformulation happens in this step and method-0' is verifiable.

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 methods proved to be verifiable. The target plan for HB2 looks like that for HB1 except that some new lemmata $L3' \ldots$ 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 as open goals the lemmata L3', L4', L5', L7', L9', L11', L17', Nit2, CIA', IA', and g'_{5a} which are left to be proved in the target.

4 Conclusion and Related Work

The presented analogy method was developed independently from the Heine-Borel example by learning lessons from the analogy examples in the standard textbook [5]. Solving the Heine-Borel analogy shows that substantial progress has been made by the analogy-driven proof-plan construction because it is able to cope with complicated real maths examples.

Our approach advances ideas of previous systems for theorem proving by analogy and of derivational analogy [4, 19]. The power of the analogy procedure partially stems from the supplied source proof-plan which guides the search for target subplans. Besides, the reformulation contributes to the strength of the analogy procedure. The remaining open goals have to be satisfied by base-level proof planning. Hence, in order for the analogy procedure to succeed, it has to be embedded into a problem solver (proof planner) which is a good idea for analogy methods ins general.

More techniques for *automatically* modifying the target methods are required such as abduction or "debugging" [2] which automatically provides additional preconditions in order to verify target methods and which has handled the proof of some difficult mathematical theorems. As also recognized in [8][9] and [9] more frequently needed reformulations have to be found.

[9] where a reformulation very similar to `Add-Argument` is used

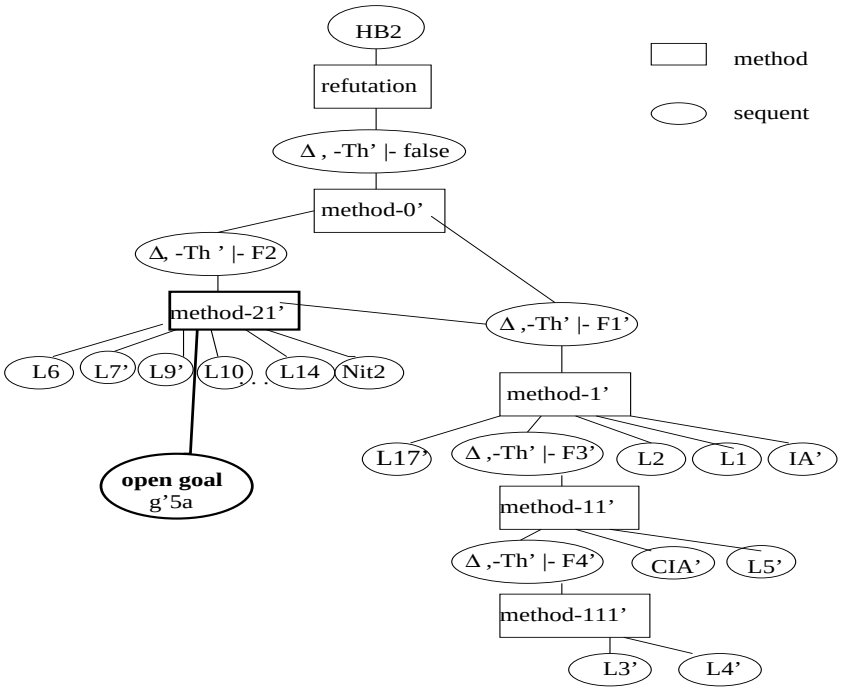

Fig. 2. The proof-plan of HB2

5 Acknowledgement

Special thanks to Woody Bledsoe who proposed a proof-plan and lemma list of HB1 [2]. He implemented a version of `Add-Argument` and also checked the validity of the plans for HB1 and HB2. Because of health problems he decided not to coauthor.

References

1. W.W. Bledsoe. The use of analogy in automatic proof discovery. Tech.Rep. AI-158-86, Microelectronics and Computer Technology Corporation, Austin, TX, 1986.
2. W.W. Bledsoe. Heine-Borel theorem analogy example. Technical Report Memo ATP 124, University of Texas Computer Science Dept, Austin, TX, August 1994.
3. 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.
4. J.G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell,

editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. Morgan Kaufmann Publ., Los Altos, 1986.
5. P. Deussen. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher*. Springer, 1971.
6. 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.
7. 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.
8. 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.
9. P. Madden. *Automated Program Transformation Through Proof Transformation*. PhD thesis, University of Edinburgh, 1991.
10. W.W. McCune. Otter 2.0 users guide. Technical Report ANL-90/9, Argonne National Laboratory, Maths and CS Division, Argonne, Illinois, 1990.
11. E. Melis. Change of representation in theorem proving by analogy. SEKI-Report SR-93-07, Universität des Saarlandes, Saarbrücken, 1993.
12. E. Melis. How mathematicians prove theorems. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 624–628, Atlanta, Georgia U.S.A., 1994.
13. 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.
14. E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the International Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.
15. E. Melis and M.M. Veloso. Analogy makes proofs feasible. In D. Aha, editor, *AAAI-94 Workshop on Case Based Reasoning*, pages 13–17, Seattle, 1994.
16. G. Polya. *How to Solve it*. 2nd ed. Doubleday, New York, 1957.
17. S.J. Russell. Analogy by similarity. In D. Helman, editor, *Analogical Reasoning*, pages 251–269. Kluwer Academic Publisher, 1988.
18. B.L. van der Waerden. Wie der Beweis der Vermutung von Baudet gefunden wurde. *Abh.Math.Sem.Univ.Hamburg*, 28, 1964.
19. 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.
20. L. Wos. *Automated Reasoning: 33 Basic Research Problems*. Prentice-Hall, Englewood Cliffs, 1988.

This article was processed using the LaTeX macro package with LLNCS style