# A Model of Analogy-Driven Proof-Plan Construction*

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

## Abstract

This paper addresses a model of analogy-driven theorem proving that is more general and cognitively more adequate than previous approaches. The model works at the level of proof-*plans*. More precisely, we consider analogy as a control strategy in proof planning that employs a source proof-plan to guide the construction of a proof-plan for the target problem. Our approach includes a reformulation of the source proof-plan. This is in accordance with the well known fact that constructing an analogy in maths often amounts to first finding the appropriate representation which brings out the similarity of two problems, i.e., finding the right concepts and the right level of abstraction. Several well known theorems were processed by our analogy-driven proof-plan construction that could not be proven analogically by previous approaches.

## 1 Introduction

Theorem proving by analogy finds a proof/proof-plan for a target problem guided by a given proof/proof-plan of a source problem which is similar to the target problem. The main use of analogy consists in reducing the search (or user interactions) by suggesting target proof steps that correspond to the source case. At the same time it can propose lemmas to be used in the target case that are similar to the source lemmas. Mathematicians have clearly recognized the power of analogical reasoning in mathematical problem solving [Hadamard, 1945; Polya, 1957]. Hence, integrating analogy into theorem provers was pointed out as one of the challenging problems in automated theorem proving in [Bledsoe, 1986; Wos, 1988].

In this paper a new approach to theorem proving by analogy is presented that works at the proof-plan level rather than at the actual proof level. First the need for a new model is substantiated by contrasting the way mathematicians use analogy to previous techniques for analogy in automated theorem proving. Then the model is presented, and finally a small example illustrates how the analogy-driven proof-plan construction works.

### 1.1 The Need for a New Model

Kling's work [Kling, 1971] was one of the first attempts in theorem proving by analogy. His system essentially produces mappings between predicate symbols. These mappings are applied to the assumptions of the source proof to find the assumptions for the target proof.

[Munyer, 1981] focusses on the formulas derived in each proof step. Munyer applies a symbol mapping, constructed from the source and target theorem, to these formulas in order to obtain the derived formulas in the target proof.

[Owen, 1990] thoroughly analyzed Kling's and Munyer's approaches to theorem proving by analogy and showed them inadequate even for many simple analogies. Owen's account emphasizes the matcher that recursively constructs symbol mappings and argument pairings. He transfers *single* calculus steps, i.e. binary resolution and paramodulation, from the source proof to provide steps in a target proof.

In [Brook *et al.*, 1988] an initial symbol mapping and a pairing of definitions and lemmas of the source with those of the target is to be provided by the user. Focussing on failed constructions of analogous proofs some heuristics are used for patching target proofs. In the same intellectual tradition Bledsoe continued dealing with the debugging of analogies by his precondition prover [Bledsoe, 1995].

In summary, computational accounts of theorem proving by analogy 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 proof steps of the source to the target[1].

Empirical investigations, however, provided evidence that this idea does not appropriately cover many analogies drawn by mathematicians, not even all the analogies of a standard textbook [Deussen, 1971] we used for an empirical study [Melis, 1993a]. I empirically analyzed mathematical theorem proving [Melis, 1994a] and found that:

*Paper at IJCAI-95

†On leave from University Saarbrücken, Germany

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

- Theorem proving by analogy is embedded into *proof planning*: Methods, rather than just single calculus steps, are transferred analogically in mathematical theorem proving by analogy.
- For many of these analogies just symbol mapping is insufficient. A change of a problem representation by, e.g., unfolding a definition, can be necessary to reveal the commonality of theorems or assumptions, upon which the analogical transfer is based. Other mappings of problems and proofs that go beyond symbol mapping are necessary as well. For some analogies, first the right level of *abstraction* has to be found before transferring a proof.
- Many proofs by analogy result from transferring *parts* of the source proof to *parts* of the target proof, and some proofs by analogy transfer only the proof idea, while others transfer a detailed proof.

## 2 The Model

Our approach takes into account the above mentioned empirical findings and incorporates

- the analogical transfer of proof-plans,
- a reformulation of proof-plans that includes abstraction[2] and other reformulations that go beyond symbol mapping.
- the restructuring of proof-plans.

Proof-plans, defined below, were introduced in [Bundy, 1988]. To motivate the analogical transfer of proof-*plans* rather than of proofs: Proof-plans are high level representations of proofs that consist of methods. We postulate that the transfer of plans by using and transforming their methods is the right level of abstraction at which to draw analogies: If methods are named, like, e.g., the Diagonalization method [Melis, 1994b], then mathematicians claim to prove the theorem by the same method. If no name is established for the way a theorem is proved, then they just say that the proof is done analogously. Proof-plans are better suited for analogical transfer than formal proofs which are often too brittle to apply a transformation in general[3]. But still proof-plans contain enough information to construct a concrete proof for a given problem. Proof-plans encode the structure of a proof because they consist of general methods encoding a proof idea and of more specific, detailed methods. Hence, analogies at different levels of details can be realized by analogically transferring to the target the transferable methods of a source proof-plan. In maths, as in all complicated domains, control knowledge, as part of mathematician's expertise, plays an important role by drastically restricting the search and by guiding problem solving. As opposed to actual proofs, proof-plans may record justifications for the proof planning decisions, e.g. the relevant control knowledge, and this information can be reused for constructing a target proof-plan. In our derivational analogy (see below) 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 [Russell, 1988] can be met. Proof-plans may record user interactions and thus avoid the tedious user-supplied guidance in theorem proving, that is usually necessary to produce a complicated proof,[4] by reusing a great deal of the previous user decisions. We integrate the call of an automated theorem prover (atp) into our plan operators and thus obtain another advantage of transferring proof-plans, namely the suggestion of proof steps likely to be feasible for an atp within a given time limit.

The description of the top level procedure will be general enough to cover analogy for different proof planners. Of course, the concrete implementation depends on the respective planner and its operators. For instance, for employing the full range of reformulations, the operator representation has to be mainly declarative. Therefore we refer to the operators designed for Ω-MKRP [Huang *et al.*, 1994a]. We briefly review how these operators and plans are defined, introduce reformulation and decomposition, and discuss the analogy procedure.

### Operators

*Sequents* $\mathsf{P} = (\Delta \vdash F)$, are pairs of a set $\Delta$ 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.

Our planning operators, called *methods*, are frame-like structures defined in [Huang *et al.*, 1994b] 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 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 automated theorem provers 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[5]. 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) for every in-

[2] Including abstraction into analogy has also been proposed in [Plaisted, 1981; Villafiorita and Sebastiani, 1994].

[3] Personal communication with Bob Harper concerning transformational tactics in Nuprl.

[4] For instance, 1741 lemmas had to be user supplied in Shankar's proof/verification of Gödel's First Incompleteness theorem[Boyer and Moore, 1988] .

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

stantiation of the meta-variables in case the constraints are satisfied. 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.

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 $g_1, \ldots, g_n$ if
$\sigma(\text{post(M)}) = g$ and $\sigma(\text{pre(M)}) = \{g_1, \ldots, g_n\}$ for a substitution $\sigma$ 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 *linearized* proof-plans ordered by the sequence in which the nodes have been added to the plan. As in [Veloso, 1994] 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.

## Reformulation

Reformulations are mappings $\rho$ 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 problem to be proved and, thus, have to be discovered empirically. Reformulations may change methods, sequents, and justifications of 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. We empirically found several classes of reformulations:

1. *Abstractions* which correspond to PI abstractions in [F.Giunchiglia and T.Walsh, 1992]. An example of an abstracting meta-method is `Homomorphism-Abstraction` (See [Melis, 1993b]). Abstractions may change methods, sequents, and justifications but preserve the subgoaling structure.
2. *Reversions* are inverse abstractions. They may change methods, sequents, and justifications but preserve the subgoaling structure.
3. *Normalizations* that add a method M and a subgoal or assumption to the proof-plan for which pre(M) $\equiv_{Th}$ post(M) for a domain theory $Th$. Usually M folds or unfolds definitions or applies logical equivalences. An example of a normalizing meta-method is `Unfold-Definitions`.
4. *Alterations* differ from abstractions and reversions. They preserve the subgoaling structure but may change methods and sequents as well as justifications. An example is `Add-Arguments` (See [Huang *et al.*, 1994b; Melis, 1995]).
5. Compositions of the reformulations above.

The sequence of reformulations that were needed to prove theorem 5.7 from the examined textbook [Deussen, 1971] by analogy is shown in Figure 1. Here the source proof-plan consists of the goal:
$\emptyset \vdash \forall f, x(f \in F \wedge x \in T_1 \rightarrow \Phi(f * x) = f * \Phi(x)))$[6] , assumptions, and one method $M_S$ only. The target goal $\emptyset \vdash \forall x, y(x, y \in H_1 \rightarrow \Phi(x \cdot y) = \Phi(x) \cdot \Phi(y))$ is theorem 5.7. The reformulation in Figure 1 aims at matching the

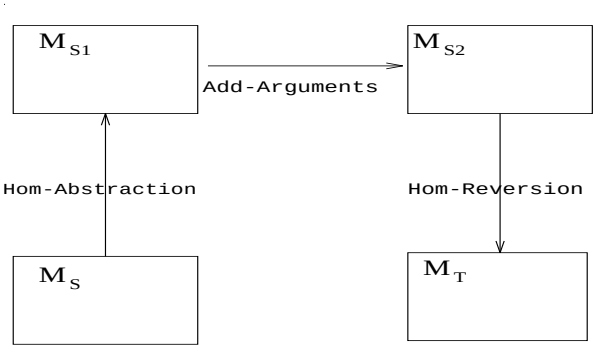


Figure 1: $M_S$ is changed to $M_T$

source goal with the target goal, and this purpose guides the reformulation of the whole source proof-plan. Starting with the source plan that contains the method $M_S$ the consecutive application of reformulations eventually provides a target plan with the method $M_T$:
$\text{post}(M_S) = (\emptyset \vdash \forall f, x(f \in F \wedge x \in T_1 \rightarrow \Phi(f * x) = f * \Phi(x)))$
$\text{post}(M_{S1}) = (\emptyset \vdash \forall x(x \in X \rightarrow \Phi(Op(x)) = Op(\Phi(x))))$
$\text{post}(M_{S2}) = (\emptyset \vdash \forall x, y(x, y \in X \rightarrow$
$\Phi(Op'(x, y)) = Op'(\Phi(x), \Phi(y))))$
$\text{post}(M_T) = (\emptyset \vdash \forall x, y(x, y \in H_1 \rightarrow \Phi(x \cdot y) = \Phi(x) \cdot \Phi(y))).$

Here `Homomorphism-Abstraction` introduces a new unary function parameter $Op$ for $f*$ and `Add-`

[6] $T_1$ denotes an $F$-semimodul, $H_1$ a semigroup

`Arguments` then replaces $Op$ by a binary function parameter $Op'$[7], while `Homomorphism-Reversion` specifies $Op'$ as $\cdot$. All these replacements require additional changes which are also realized by the mentioned meta-methods. For the detailed reformulation see [Melis, 1993b].

### Decomposition

In addition to reformulation, the analogy-driven proof-plan construction decomposes methods of proof-plans as well. Decomposition maps a proof-plan to a proof-plan by replacing a sub-plan $\mathcal{P}$ with one method by a sub-plan with several methods while preserving the root and leaves of $\mathcal{P}$. An example is `Conjunctive-Decomposition` that decomposes a method M with post(M) $= (\Delta \vdash F_1 \wedge F_2)$ into a plan with methods $M_1$, $M_2$, and $M_\wedge$ with post($M_1$) $= (\Delta \vdash F_1)$ and post($M_2$) $= (\Delta \vdash F_2)$.

Decomposition is applied at two different places in the analogy procedure for two different purposes: Firstly, decomposing a source method yields a subgoal that matches an open target goal. This is the case if only a part of the source plan can be transferred and if the source method contains a transferrable as well as a not transferrable part. Secondly, decomposition can be used if a reformulated method cannot be verified. This method is decomposed in order to find a verifiable submethod.

## 2.1 Analogy-Driven Proof-Plan Construction

Problem solving by analogy transfers the way a problem is solved (derivational analogy) or the final solution (transformational analogy). Derivational analogy in [Carbonell, 1986; Veloso, 1994] stores problem solving decisions and their justifications in the source plan and replays them for the target. Transformational analogy in [Carbonell, 1983] takes the source solution and transforms it into the target solution.

Our analogy-driven proof-plan construction can be considered a combination of derivational with transformational analogy.[8] Basically it is a derivational analogy, however, it also transforms the source plan. Actually, it is a control strategy for proof planning that extends the derivational analogy of [Veloso, 1994] by reformulation, decomposition, and bidirectional planning. The general idea of our analogy model is to use the linearized 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.

[7] `Add-Arguments` is applicable to a proof-plan if a function $f$ with $n$ arguments is to be mapped to an $f'$ with $n+m$ arguments, where the $m$ arguments to be duplicated are specified out of the $n$ arguments of $f$ as parameters of the meta-method. The meta-method replaces the $f$ by $f'$ in goals, assumptions, and methods and yields additional related changes in methods.

[8] For example, "initial-segment concatenation" and "final-segment concatenation" in transformational analogy correspond to some normalizing reformulations.

Table 1 shows the top-level procedure of our analogy-driven proof-plan construction. Given a parametrized, linearized source proof-plan, target assumptions, and a target goal (the first open goal), the output of the procedure is a target proof-plan. Steps 5 - 7 are those that are relevant for a planner with backward search only. Steps 8-10 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. $|missing(\rho' \mathrm{M})| < n$ means[9] that less than $n$ preconditions of the currently treated reformulated f-method M do not match a 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 8.
4. **if** P is already satisfied, **then** go to 2.
5. **if** there is a reformulation $\rho$ left, such that $\rho$P matches an open target goal $g_T$ for which the justifications hold,
   - **then** reformulate source plan by $\rho$ and link $g_T$ to source plan.
   - **else** decompose source and go to 2.
6. Select for the target the b-method M chosen in reformulated source.
7. **if** M's justifications hold for in the target,
   - **then** verify/modify M and go to 1.
   - **else** go to 5
8. Select for the target the f-method M chosen for P in the source.
9. **if** there is a reformulation $\rho'$ left such that $|missing(\rho' \mathrm{M})| < n$ and that justifications hold for the matched target assumptions
   - **then** reformulate source plan by (best) $\rho'$ and link the matched target assumptions to source plan.
   - **else** decompose source and go to 2.
10. **if** M's justifications hold for $\rho'$M in the target,
    - **then** verify/modify $\rho'$M and go to 1.
    - **else** go to 9.

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

The first goal of the linearized source plan, usually the source problem $\mathsf{P}_S$, is chosen in 3. If $\mathsf{P}_S$ can be reformulated by a $\rho$ such that it matches the target problem $\mathsf{P}_T$, then $\rho$ 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. If $\mathsf{P}_S$ cannot be reformulated by a $\rho$ to $\mathsf{P}_T$, then decomposition is applied in order

[9] For *missing(M)* := set of preconditions of M that do not match a current target assumption. The sequents of $missing(\rho' \mathrm{M})$ become new open goals if $\rho'$ is an acceptable reformulation.

to find a submethod $M_1$ of M that can be transferred. If in 5 or 9 a decomposition is not possible, then the plan stays unchanged. After the decomposition a new reformulation $\rho$ and match of post($M_1$) is tried and 6. suggests $\rho M_1$ as a candidate a target method. 7. checks whether the justifications of the source method hold in the target for $\rho M_1$ and if so, its verifiability is checked. The latter test is necessary because reformulation does not necessarily preserve verifiability. If $\rho M_1$ is not verifiable, then a promising modification is applied (either decomposing the method in order to obtain a verifiable submethod or calculating additional preconditions of the method yielding verifiability.). In table 1 verify/modify M abbreviates:

- test verifiability of M
- **if** M is not verified, **then** modify M to a verifiable (sub)method[10]
- update open target goals and assumptions
- link the verifiable method to source plan.

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. Superfluous steps in the target plan are skipped in 4. This procedure yields a target plan with verified methods. The target plan may have open goals.

Phrased in the terminology of case-based reasoning, adaptation takes place via reformulation of proof-plans, through modifying not verifiable methods, skipping superfluous methods, and through closing gaps in the target plan by base-level planning.

The possible objection that every new example would need a new set of meta-method did not turn out to be true. Reformulation focussed on alteration and, unexpectedly, `Restricted Term-Mapping` and `Add-Argument` were used quite often.

### Controlling the Search for Reformulations

So far we have only few meta-methods available and thus search is restricted. As the set of meta-methods expands, however, the following control features become more important:

- The application of meta-methods is controlled by their application-conditions.
- The reformulation of source plans is guided by the target goals and assumptions to be matched.
- The following fixed sequence of reformulations proved most useful: normalization, abstraction, alteration, reversion and should thus be superimposed upon search.
- User-supplied correspondence tables for relations and functions in specific maths areas carry semantic information and provide additional guidance for reformulations. A correspondence table helps to relate source to target assumptions and may prescribe the mapping of function or relation symbols.

With a growing variety of reformulations it will be necessary to meta-plan the application of meta-methods using the application-conditions and effect or to interactively choose reformulations.

[10] No result if no verifiable (sub)method exists.

## 3 Example

An example that illustrates all features of the approach (as, e.g., in [Melis, 1995]) needs too much space. The following example does not take much advantage of proof-plans; it focusses on decomposition and uses the meta-methods `Term Mapping` and `Symbol Mapping` only.
Source (sub)goals:
gS = $even(a) \wedge even(b) \rightarrow even(a*b)$,
g1S = $even(a) \wedge even(b) \vdash a*b = 2*c$,
assumptions:
a1S = $\emptyset \vdash \forall x(even(x) \rightarrow \exists y(x = 2*y))$ and
a2S = $\forall x \forall y \forall z((x*y)*z = x*(y*z))$
Target goal:
gT = $odd(a) \wedge odd(b) \vdash odd(a*b)$
assumptions:
a1T = $\emptyset \vdash \forall x(odd(x) \rightarrow \exists y(x = 2*y+1))$ and
a2T = $\forall x \forall y \forall z((x*y)*z = x*(y*z))$.
Figure 2 shows how a target proof-plan is constructed incrementally guided by the source proof-plan:

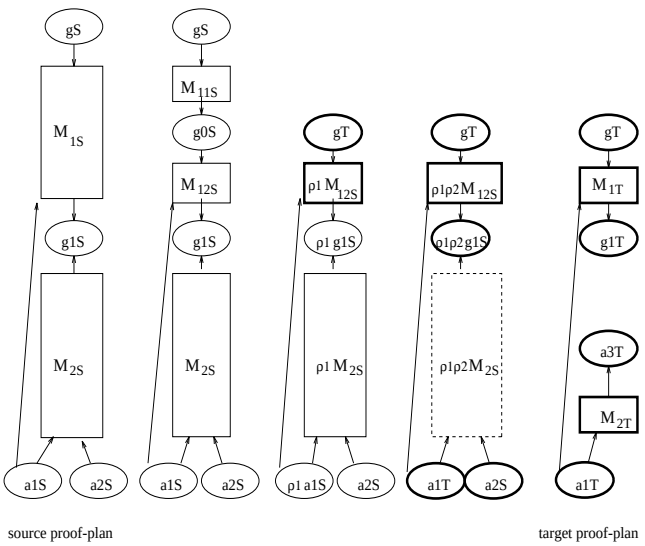


Figure 2: Analogy-driven proof-plan construction

- Method $M_{1S}$ is decomposed into $M_{11S}$ and $M_{12S}$ because gS can not be reformulated such that it matches gT.
- g0S = post($M_{12S}$) = $even(a) \wedge even(b) \vdash even(a*b)$ as well as the proof-plan is reformulated by the `Symbol Mapping` $\rho_1$: ($even \Rightarrow odd$) yielding $\rho_1$g0S = gT.
- $\rho_1$a1S and the whole plan is further reformulated by the `Term Mapping` $\rho_2$: ($2*t \Rightarrow 2*t+1$) in order to match the target assumption a1T. The resulting method $\rho_1\rho_2 M_{12S}$ is verifiable and has the precondition $\rho_1\rho_2$g1S = g1T = $odd(a) \wedge odd(b) \vdash \exists c(a*b = 2*c+1)$.
- The method $\rho_1\rho_2 M_{2S}$ is not verifiable and, hence, is decomposed into $\rho_1\rho_2 M_{21S}$ and $\rho_1\rho_2 M_{22S}$, where $\rho_1\rho_2 M_{22S}$ becomes $M_{2T}$ and

post($M_{2T}$) = a3T = $odd(a) \wedge odd(b) \vdash$
$a * b = (2 * a_1 + 1) * (2 * b_1 + 1)$.

- Since the source plan is exhausted now, the gap between the assumption a3T and the open subgoal g1T has to be closed by base-level planning for g1T using a1T, a1S and maybe additional lemmas.

## 4 Conclusion and Future Work

Analogy in theorem proving does not work as a stand alone problem solver but works very much in connection with ordinary theorem proving, for instance, to validate details which are left as open goals. Besides, a source proof-plan has to be provided using proof planning. There are neither guaranteed success nor completeness claims in this context because analogy is a heuristic strategy that is tentative.

Our approach combines ideas of previous systems for theorem proving by analogy with derivational and transformational analogy. It works in a planning framework and incorporates reformulation, and thus, it is more widely applicable than previous approaches which did not attempt to re-represent the source problem and, hence, their results were highly dependent on the actual representation of theorems and proof assumptions. Wider applicability is documented by several experiments with the analogy-driven proof-plan construction on *real* mathematical theorems which succeeded in providing analogous proofs where other approaches did not. The experiments dealt with theorems from a standard text [Melis, 1993b], the pumping lemma for context free languages [Melis and Veloso, 1994], and a Heine-Borel theorem [Melis, 1995]. These theorems provide a small but quite a representative sample. Furthermore as discussed above, the approach is cognitively more adequat: Among others, [Melis, 1995] has shown that the interpretation of parameters, that might be necessary for completing an analogous proof, corresponds to situations observable for a mathematician's use of analogy.

In general there will be a tradeoff between the higher flexibility and the search needed for reformulation in automatic analogy-driven proof-plan construction. Since usually only few reformulations are needed, however, the search for appropriate reformulations is much less than the search usually needed for a proof.

Currently some reformulations are implemented but chosen by the user. The retrieval of source proof-plans has not yet been approached. Hence, up to now we model common situations in which the source is given only. Besides further implementation future work is necessary to retrieve the source proof-plan automatically, to automatically control the application of reformulations, and to discover more, frequently used maths reformulations.

## Acknowledgements

Many thanks to Woody Bledsoe for help and encouragement. I would like to thank Manuela Veloso, Manfred Kerber, Xiaorong Huang, Dieter Hutter, and Alan Smaill.
This work was supported by a research grant of the Deutsche Forschungsgemeinschaft and HC&M contract CHBICT930806.

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

[Boyer and Moore, 1988] R.S. Boyer and J.S. Moore. *A Computational Logic Handbook*. Academic Press, San Diego, 1988.

[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, 1983] J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. TiogaPubl., Palo Alto, 1983.

[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.

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

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

[Hadamard, 1945] J. Hadamard. *The Psychology of Invention in the Mathematical Field*. Princeton Univ. Press, Princeton, 1945.

[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.

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

[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 and Veloso, 1994] 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.

[Melis, 1993a] E. Melis. Analogies between proofs – a case study. SEKI-Report SR-93-12, Universität des Saarlandes, Saarbrücken, 1993.

[Melis, 1993b] E. Melis. Change of representation in theorem proving by analogy. SEKI-Report SR-93-07, Universität des Saarlandes, Saarbrücken, 1993.

[Melis, 1994a] 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.

[Melis, 1994b] E. Melis. Representing and transferring diagonalization methods. Technical Report CMU-CS-94-174, Carnegie Mellon University, Dept. of Computer Science, Pittsburgh, Pennsylvania, U.S.A., 1994.

[Melis, 1995] 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.

[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.

[Russell, 1988] S.J. Russell. Analogy by similarity. In D. Helman, editor, *Analogical Reasoning*, pages 251–269. Kluwer Academic Publisher, 1988.

[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.