AI-Techniques in Proof Planning

Erica Melis[1]

Abstract. Proof planning is an application of Artificial Intelligence (AI)-planning in mathematical domains for theorem proving. The paper presents a knowledge-based proof planning approach that is implemented in the OMEGA proof planner. It evaluates control-rules in order to restrict the otherwise intractable search spaces and combines proof planning with domain-specific constraint solving. Several AI-techniques contribute to the successful planning of proofs that were beyond the capabilities of theorem provers and proof planners previously.

1 Introduction

Classical logic-based automated theorem provers, such as OTTER [16], have gained considerable strength. They could even prove some non-trivial open mathematical problems such as the Robbins algebra conjecture [15] whose proofs are rather unintuitive and tricky for humans. In general, however, most proofs of genuinely mathematical problems are beyond the capabilities of these theorem provers because traditional automated theorem proving that works with calculus-level inference steps, e.g. resolution, suffers from exhaustive search in super-exponential search spaces[2] and does not have a 'sense of direction'. Moreover, these systems produce calculus-level proofs that are difficult to understand even for experts. Proof planning can help avoiding both problems.

There is a *cognitive* motivation for proof planning too that is due to the fact that mathematicians tend to plan proofs. Several sources [1, 14] provide empirical evidence that mathematicians use specific methods (e.g., diagonalization) and plan a proof during the proof discovery process. Although this observation seems to be obvious for everyone who has been engaged in mathematics, the question is, how to realize this technically and how to represent the knowledge needed. These questions are addressed in the paper.

In order to avoid super-exponential search, Bundy [6] introduced proof planning for proving theorems by induction. This proof planning for inductive proofs is implemented in the proof planner CL^AM[8]. It employs a global search heuristic for difference reduction, rippling, and its operators are complex inferences such as `induction`, `base-case`, and `step-case` rather than calculus-level rules. While rippling suffices for proofs working by difference reduction, it fails to guide other mathematical proofs. For instance, the limit theorem LIM* was beyond the capabilities of current theorem provers and proof planners including CL^AM.

In order to cope with genuinely mathematical problems we extended the proof planning methodology such that proof planning can use domain knowledge.

Contribution of the Paper. In his invited talk at IJCAI-97 [3], Bibel suggested to solve many AI-problems by automated theorem provers. Vice versa we suggest to employ AI-techniques for automated proof planning. In particular, we show how we have made progress in proving mathematical theorems a knowledge-based proof planning approach and by employing AI-techniques that restrict the search.

The paper presents the general knowledge-based proof planning. Knowledge that is used in planning proofs for a class of limit theorems illustrates our approach. We conclude with results and an evaluation.

2 Brief Review of Proof Planning

The basic idea of proof planning is that of AI-planning [9]. Here, operators, called *methods*, encode mathematical proof methods. These methods represent inference actions that usually are more complex than the inference steps in a logical calculus and therefore the resulting plans are abstract representations of proofs.

Technically, methods are (partial) specifications of tactics originating from tactical theorem proving [10]. Tactics are procedures that execute several inference steps. In OMEGA [2] the methods are represented by frame-like structures. Their specifications used in planning are provided by the slots *premises* and *conclusions* that contain annotated proof lines. The *application-conditions* restrict the method's application. *proof schema* may contain a schematic partial plan whose instantiation is introduced into the plan when the method is expanded. The line-justifications (right most entry) of the proof lines in *proof schema* are methods (including tactics) or OPEN, see the examples of methods below.

In OMEGA, a planning state contains a set of sequents[3] represented by proof lines that is divided into "open lines" and "closed lines". A proof planning problem consists of a theorem to be proved and of the proof assumptions. A problem defines an initial state specified by closed lines (the proof assumptions, axioms, definitions) and by an open line provided by the theorem to be proved.

The annotations of *premises* and *conclusions* indicate the state changing potential of a method: a $\ominus$ conclusion is deleted as an open line, a $\oplus$ premise is added as a new open line

[1] Universität des Saarlandes, Fachbereich Informatik, D-66041 Saarbrücken, Germany

[2] Up to three billion clauses clauses are generated. Personal communication with Bill McCune.

[3] A sequent is a pair $(\Delta \vdash F)$ with a set of formulae Δ and a formula F. Its meaning is F is derived from Δ.

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

(backward planning), a $\ominus$ premise is deleted as an closed line, and a $\oplus$ conclusion is added as an closed line.

The planner searches for a solution of the problem, i.e., a sequence of instantiated methods that transforms the initial state into a state with no open lines. When a solution is found, its instantiated methods can be recursively expanded to a Natural Deduction (ND)-proof.

Since methods are more complex than calculus-level inference rules, proof planning searches at a more abstract level than classical theorem proving does. However, at this level the problem of prohibitively large search spaces occurs again because a potentially infinite branching of the search space can occur when existentially quantified variables have to be instantiated or when a lemma has to be introduced. Hence techniques for reducing the search are needed in proof planning.

3 Knowledge-Based Proof Planning

A reduction of the search can be achieved by making domain knowledge accessible to a general-purpose problem solver. Our knowledge-based proof planning can use domain knowledge organized in a hierarchical theory knowledge base containing domains such as *base*, *group*, and *dense-linear-order*. In addition to the traditional content of theories (axioms, theorems, and definitions), our domain knowledge comprises methods, control-rules, and domain-specific constraint solvers. The extended knowledge base is depicted in Figure 1. The figure

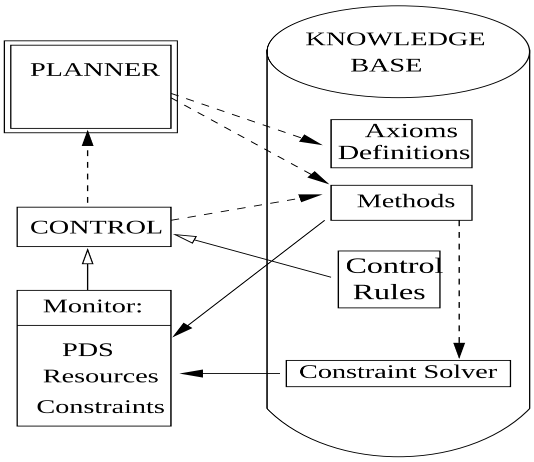

Figure 1. Architecture of the planner

summarizes the architecture required for the knowledge-based proof planning: a control unit evaluates control-rules. The evaluation of control-rules may inspect the proof plan data structure (PDS) including its history, the planning state, resources, and the constraint state. This evaluation yields a restriction of the alternative choices at each choice point of the planner, e.g., for the choice of goals and methods. From the restricted set of alternative methods the planner chooses a method and introduces it into the PDS. A theory-specific constraint solver can be accessed via methods and it can change the constraint state. Furthermore, it can provide an answer constraint formula that instantiates the meta-variable introduced for the constraint state of a constraint solver.

Below we explain methods, control-rules, and the integration of a constraint solver in more detail and we illustrate this knowledge-based proof planning with the class of limit theorems. A well known limit theorem, LIM+ (LIM*), states that the limit of the sum (product) of two functions is the sum (the product) of their limits. Bledsoe [4] introduced LIM+ as a challenge problem. LIM+ is at the edge of what other theorem proving systems or proof planners can prove, and certainly LIM* is beyond their capabilities.

3.1 Methods

The methods that belong to the domain knowledge represent frequently used mathematical proof steps. In the mathematical literature, some methods have names, such as diagonalization or induction. However, many mathematical methods do not have a name and the knowledge engineering for methods that can be used in a number of plans turned out to be complicated and to require thorough mathematical knowledge.

In proving limit theorems, estimations of term magnitudes are needed frequently. Hence we designed estimation methods. To alleviate the burden of knowledge engineering for one of the proof methods for limit theorems, we rationally reconstructed Bledsoe's limit heuristic that was used in a special-purpose program [5]. In a similar way, the procedures of other special-purpose theorem provers can be reconstructed.

The resulting `ComplexEstimate`$_<$ is a method for estimating the magnitude of the absolute value of complex terms such as $(f(x) - g(x)) - (L_1 - L_2)$.

method: `ComplexEstimate`$_<$ (a, b, e_1, ϵ)				
premises	(0), $\oplus$(1),$\oplus$ (2), $\oplus$(3)			
conclusions	$\ominus th$			
appl.cond	$\exists k, l, \sigma(extract(a, b) = (k, l, \sigma))$			
proof schema	(0). Δ	$\vdash$	$\lvert a\rvert < e_1$	()
	(1). Δ	$\vdash$	$\lvert k_\sigma\rvert < \mathbf{M}$	(OPEN)
	(2).	$\vdash$	$\lvert a_\sigma\rvert < \epsilon/2 * \mathbf{M}$	(OPEN)
	(3). Δ	$\vdash$	$\lvert l_\sigma\rvert < \epsilon/2$	(OPEN)
	l1.	$\vdash$	$b = k_\sigma * a_\sigma + l_\sigma$	(CAS)
	l2.	$\vdash$	$0 < M$	(OPEN)
	th. Δ	$\vdash$	$\lvert b\rvert < \epsilon$	(fix;l1, (1),(2),(3))

The planner handles `ComplexEstimate`$_<$ as follows.

- If an open line in the planning state matches *th* and if an assumption matching (0) is available, `ComplesEstimate`$_<$'s parameters a, b, e_1, ϵ are instantiated by the matcher.
- Then *application-conditions* is evaluated. That is, the procedure *extract*(a, b) is invoked. *extract* is an oracle function that takes the input terms a, b and may return a list (k, l, σ) for terms k, l and a substitution σ such that $b = k_\sigma * a_\sigma + l_\sigma$.[4] If *extract* returns a list (k, l, σ), then the method is applicable and the variables $a_\sigma, k_\sigma, l_\sigma$ are bound to terms resulting from applications of σ to a, k, l.
- If the *application-conditions* evaluate to *true*, `ComplexEstimate`$_<$ is introduced into the partial plan, the open line *th* is removed, and the new open lines (1), (2), and (3) are introduced instead.
- When the method is expanded later on, the *proof schema* is used. It contains a schematic proof of $\Delta \vdash |b| < \epsilon$ from $b =$

[4] $/, *, ||$ denote the division, multiplication, and absolute value functions, σ denotes a substitution, and F_σ the result of applying σ to F.

$k_\sigma * a_\sigma + l_\sigma$, (0), (1), (2), and (3). The fixed subproof "fix" employs, among others, the triangle inequality $|X + Y| \leq |X|+|Y|$. The line-justification CAS names a tactic that can justify the equation $b = k_\sigma * a_\sigma + l_\sigma$. During the expansion of `ComplexEstimate<`, the tactic runs and returns a proof plan for its computation.

For instance, in planning LIM+, the assumption $\Delta \vdash |f(X_1) - l_1| < E_1$ is available at some point. The input of `ComplexEstimate<` is the open line $\Delta \vdash |f(x) + g(x) - (l_1 + l_2)| < \epsilon$. When the *application-conditions* are evaluated, *extract* returns the list $(1, (g(X_2) - l_2), [x/X_2])$ since a is instantiated by $(g(X_2) - l_2)$ and b is instantiated by $(f(x) + g(x) - (l_1 + l_2))$. Then the open line $\Delta \vdash |f(x) + g(x) - (l_1 + l_2)| < \epsilon$ is replaced by the new open lines

1. $\Delta \vdash |1| < \mathbf{M}$,
2. $\Delta \vdash |f(X_1) - l_1| < \epsilon/2 * \mathbf{M}$,
3. $\Delta \vdash |g(x) - l_2| < \epsilon/2$.

Each application of `ComplexEstimate<` suggests the existence of a real number $\mathbf{M}$ whose range is restricted by the inequalities (1) and (2). Note that $\mathbf{M}$ is a meta-variable[5] that is used to propagate given range restrictions from universally quantified variables such as ϵ to meta-variables such as D as described section 3.3.

Note that with `ComplexEstimate<` as with many other methods, the theorems and axioms from the domain theory, such as the triangle inequality or the transitivity of $<$, do not need to be referred to explicitly but are invisible at the planning level. Only when the methods are expanded to a lower level, the detailed *proof schema* is introduced into the proof plan and so is the explicit application of axioms. This is a way to avoid the situation common in classical automated theorem proving that requires to provide exactly those axioms and definitions needed in the particular proof because otherwise the system gets lost with too many applicable axioms.

The following estimation method `Solve<b` can be applied to inequalities with simple terms. `Solve<b`, provides an interface to a constraint solver for linear inequalities in $\mathcal{R}$, described in section 3.3.

method: `Solve<b`(a, b)	
premises	L1
conclusions	$\ominus$L2
appl.cond	**if** var-in$(a < b)$ **then** *tell*$(a < b) = true$ **else** *ask*$(a < b) = true$
proof schema	L1. $\Delta \vdash CS$ $()$ L2. $\Delta \vdash (a < b)$ (solv(CS))

In addition, this method can be parametrized by the theory that underlies the particular constraint solver to be used. In the following, we consider a constraint solver for *linear arithmetic* in $\mathcal{R}$ only.

The operator `Solve<b` is applied to an open line L2 with an inequality $(a < b)$. `Solve<b` removes this open line from the planning state. Since the method `Solve<b` has no preconditions, it produces no new open lines. The *application-conditions* decide which of the functions *tell* or *ask* is chosen as an interface to the constraint solver. *tell* is chosen if $(a < b)$ contains a meta-variable and *ask* is chosen for inequalities with implicitly universally quantified variables and constants only. The function *tell* tells an inequality c to the constraint solver which checks the consistency of c with the constraint store. If c is consistent with the constraint store, *tell* returns $true$. *ask*(c) asks the constraint solver whether c is entailed by the constraint store; if it is, *ask* returns $true$. This distinction is necessary because the value of universally quantified variables and constants cannot be restricted without loss of generality whereas the value of meta-variables representing existentially quantified variables can.

In *proof schema*, the line-justification solv(CS) names a tactic that can justify $(a < b)$ from the final constraint store that is condensed to a formula cs at the end of the proof planning. For each constraint solver i the corresponding formula cs_i instantiates a meta-variable CS_i that has been introduced during the planning and that occurs in the proof plan.

3.2 Mathematical Control Knowledge

For automated proof planning, control knowledge is needed in addition to the methods. Some of the general control knowledge belongs to *base*, e.g., the knowledge on when to introduce a case split. Since proof planning is performed at a level that is more abstract than the logic calculus level, it is possible to employ more abstract control knowledge, that is, mathematical control knowledge. This kind of control knowledge is better to communicate and easier to understand than the usual parameter settings in classical automated theorem proving.

In OMEGA, control-rules encode this mathematical control knowledge. At each decision point in planning the evaluation of these control-rules suggest preferred or selected methods, i.e., it reduces the number of alternative methods to be considered by the planner or of open lines to be considered next. The control-rules are interpreted before matching the methods' specifications with the lines of the current planning state and this can reduce the set of methods to be matched.

We extracted mathematical control knowledge for planning limit theorems and translated it to control-rules. Some mathematical knowledge on how to proceed reads as follows.

1. Linear inequalities can be proved by direct estimation or by a decomposition that introduces auxiliary variables.
2. For such a decomposition proof, a (part of a) proof assumption has to be used.

The mathematical knowledge can be translated into the following verbally expressed control knowledge that refers to proof planning methods.

1. Open lines with linear inequalities can be removed by `Solve<b`, `SOLVE*`, or by `ComplexEstimate<`.
2. The latter requires some preparation by `UNWRAPHYP`. `UNWRAPHYP` extracts a subformula s from an assumption in order to afterwards employ s by `ComplexEstimate<`.

Now this knowledge can be formally represented by control-rules, for instance, by

```
(control-rule prove-inequlities
      (if (goal-matches ("goal" ("x < y"))))
```

[5] Meta-variables are place holders for instantiations of existentially quantified variables.

```
(then (prefer ((Solve<b "goal")
               (SOLVE* "goal")
               (UNWRAPHyp "goal")))))
```

A small collection of control-rules that includes the presented one led to a successful planning of all the limit theorems and is presented in [18].

3.3 Constraint Solving in Proof Planning

In many mathematical proofs, logical steps are combined with specialized reasoning such as computing integrals, solving polynomial equations, and solving inequalities. Usually, a purely logical proof is possible in principle but far too long and inefficient when a construction of mathematical objects with certain properties is needed, e.g., the instantiation of the existentially quantified δ in LIM+. For such proofs a purely logic-based system may generate a search space that could easily prevent the system from finding a proof.

Some search can be avoided by delaying the instantiation of meta-variables through the incremental restriction of their values and by reducing the space of possible instantiations with the help of domain-specific constraint solving. Constraint solvers can check consistency and propagate value restrictions of variables very efficiently because they provide efficient data structures and algorithms for specific domains, e.g., for finite domains [11], for sets [20], and for linear arithmetic in $\mathcal{R}$ [13].

At the planning level, a domain-specific constraint solver that is integrated into a proof planning framework serves two main purposes: Firstly, it is used during the process of proof planning to determine whether certain methods, such as $\texttt{Solve}_<\texttt{b}$, can be legally applied. Secondly, after the completion of the proof plan, the constraint store is condensed into an answer assertion cs_i about the values of meta-variables. This assertion is included into the final proof by instantiating the meta-variable CS_i that is originally introduced for each constraint solver$_i$ by the `Choice` method. Moreover, this answer assertion cs_i justifies the constraints that were removed as open lines during planning.

Interface.

Test functions interface the planner with a constraint solver via methods. As described above, through the *application conditions* the methods decide whether to access the constraint solver via *tell* or via *ask*. *tell* checks constraints c for consistency with the current constraint store and propagates c in case of consistency. *ask* checks for entailment of a constraint from the constraint store. For planning proofs of limit theorems our most important methods at the interface between the proof planning and the constraint solver are $\texttt{Solve}_<\texttt{b}$ and $\texttt{Solve*}_<\texttt{b}$. Their purpose is to remove open lines with an inequality by adding this constraint to the constraint store as long as the constraint is consistent with the constraint store.

In addition, an *instantiation function* delivers the instantiation of the meta-variable CS_i to the planner. An *expansion functions* provides a trace of the constraint solver's activity needed to expand the proof plan to a ND-level proof if this is desired.

The control is on the planner side only. In particular, the choice of methods, including `Solve` methods, the *ask/tell* interfacing, the expansion of methods, and the introduction and instantiation of meta-variables.

Example.

When planning the proof of the LIM+ theorem, the following constraints are told to the constraint solver in a row: $0 < D, 1 < M, E_1 < \epsilon/2 * M, x = X_1, E_2 < \epsilon/2, x = X_2, D < \delta_2, D < \delta_1$, where $D, E_1, E_2, \delta_1, \delta_2, \epsilon$ stem from the planning problem and M is the auxiliary meta-variable introduced by $\texttt{ComplexEstimate}_<$. From the constraints $1 < M$ and $E_1 < \epsilon/2 * M$ the new upper bound $\epsilon/2$ of E_1 is propagated. This leads to a final constraint store that looks as follows

$$
\begin{aligned}
&0 < E_2 < \epsilon/2;\\
&0 < D < \delta_2(E_2), \delta_1(E_1);\\
&0 < E_1 < \epsilon/(2 * M), \epsilon/2;\\
&1 < M < \epsilon/(2 * E_1)\\
&-\infty < X_1 = x = X_2 < +\infty
\end{aligned}
$$

These accumulated value restrictions can be projected to an answer assertion $\epsilon_1 < \epsilon/2 \wedge \epsilon_2 < \epsilon/2 \wedge \delta < \delta_1 \wedge \delta < \delta_2$ that reads "Let $\epsilon_1 < \epsilon/2, \delta < \delta_1, \delta_2 \ldots$" as known from proofs in mathematics textbooks. The projection removes redundancies and replaces as many bounds as possible by numeric bounds.

4 Results and Conclusion

Proof planning is AI-planning with operators that represent mathematical methods. Compared to classical automated theorem proving, proof planning produces proof plans that are abstract representations of proofs. These proof plans provide a basis for comprehensible proof presentation and for other tasks such as analogy-driven proof plan construction [17].

This paper emphasized a knowledge-based proof planning approach that allows to combine the general proof planning with domain knowledge comprising methods and control-rules. The evaluation of control-rules reduces the search for methods. The paper briefly investigated the combination of (proof) planning with constraint solving in order to restrict the search. The proof planning paradigm can take advantage of classical automated theorem provers too by using them as tactics that occur in the justifications of (trivial) proof lines.

Based on AI-planning experience we have gained a deeper understanding of proof planning and have extended its methodology and the applicability of proof planning to mathematical domains. Vice versa, proof planning led to new ideas for AI-planning such as means to properly handle infinite domains by combining planning and constraint-solving (for the instantiation of existentially quantified variables). Other contributions to AI-planning are the introduction of supermethods and multi-strategy planning not discussed in this paper.

Results.

We illustrated the approach by presenting knowledge that is needed in order to plan all limit theorems. We succeeded in planning these theorems among them theorems that were beyond the problem solving horizon of other systems, e.g. LIM*. Other theorems we proved are, e.g., ContinuousComp that states that the composition of two continous functions is continuous, and ContinuousIfDeriv that states that a function having a derivative at a point is continuous there. The use of control-rules was essential for this success. The constraint solving component helped restricting the instantiations of existentially quantified variables. The results show that our approach is useful.

A comparison of the search spaces of proof planning and classical automated theorem proving, epitomized by OTTER [16], shows a qualitative progress: For a simpler version of LIM+ and with a particular strategy and parameters, OTTER generates a search space of 3650 clauses and keeps 1418 clauses.[6] However, this strategy does not work for LIM* or other limit theorems. In auto-mode OTTER generates up to 437706898 clauses and does not succeed in proving LIM+. In comparison, the search space of our proof planning for LIM+ is 350 nodes.[7] The control-rules are necessary to find a plan for LIM+. However, with the same methods and control-rules OMEGA's planner succeeds to prove all the limit theorems. For instance, the search space of planning LIM* has 470 nodes.

Evaluation.

How can the work presented be assessed in terms of of the criteria for proof plans from Bundy's Science of Reasoning [7]? *Expectancy*: The methods and control work for planning the limit theorems. *Generality*: All methods can be used for a larger class of proof plans. The `Solve` methods can even be generalized beyond the particular constraint solver for linear inequalities, e.g., in set theory another constraint solver has to be integrated. No fixed structure of limit plans can be established but common control-rules. *Prescriptiveness*: The expansions of methods does not produce search and the plans are not over-detailed. *Simplicity*: From the planning point of view, the specifications is simple. The knowledge engineering process for designing the methods was not simple though. *Efficiency*: The described proof planning avoids search and is efficient. *Parsimony*: We need several control-rules for the collection of all limit theorems. These are, however, not specific for one theorem.

The knowledge-based proof planning has several advantages over the procedurally encoded use of specific knowledge in special-purpose theorem provers, e.g., for set theory [12]: it employs a general-purpose problem solver, the plans are expandable to checkable calculus-level proofs, the control knowledge is modular and can therefore be subject to modification, and last but not least, the resulting plans can be communicated to the user.

A bottleneck of the knowledge-based proof planning is the knowledge acquisition. In mathematics the acquisition of methods and control knowledge is more difficult than in typical AI-planning domains, e.g., the transportation domain or blocks world. Often the mathematical knowledge about methods and control in textbooks etc. is implicit only and it is difficult to make it explicit. This is one reason why mathematics is hard for many people.

Related Work.

Although closest to our work, proof planning in *CLAM* is different in several aspects. As opposed to our domain-dependent control knowledge, *CLAM* uses the general difference reduction heuristic rippling only. *CLAM* does not integrate constraint solving. As opposed to our hiding of axioms, *CLAM* uses axioms and definitions explicitly. As far as we know, LIM* has not be proved by *CLAM*. However, LIM+ was proved in *CLAM* with colored rippling [21], an extension of rippling.

Our work is related to AI-planning, e.g. in Prodigy [19] that uses control-rules. Moreover, it is related to theorem proving approaches that integrate constraint solving [20] or that try to use domain knowledge [12] in special-purpose provers.

[6] Specific input file of Bill McCune.

[7] This figure applies for a plan that includes the elimination of quantifiers. It can still be optimized.

REFERENCES

[1] J.R. Anderson, 'Acquisition of proof skills in geometry', in *Machine Learning, An Artificial Intelligence Approach*, eds., R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, 191–219, Springer, (1984).

[2] C. Benzmueller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, and V. Sorge, 'OMEGA: Towards a mathematical assistant', in *CADE-14*, pp. 252–255. Springer, (1997).

[3] W. Bibel, 'Let's plan it deductively', in *IJCAI-97*, pp. 1549–1562, (1997).

[4] W.W. Bledsoe, 'Challenge problems in elementary analysis', *Journal of Automated Reasoning*, **6**, (1990).

[5] W.W. Bledsoe, R.S. Boyer, and W.H. Henneman, 'Computer proofs of limit theorems', *Artificial Intelligence*, **3**(1), 27–60, (1972).

[6] A. Bundy, 'The use of explicit plans to guide inductive proofs', in *CADE-9*, pp. 111–120. Springer, (1988).

[7] A. Bundy, 'A science of reasoning', in *Computational logic: Essays in Honor of Alan Robinson*, eds., J-L. Lassez and G. Plotkin, 178–198, MIT Press, (1991).

[8] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill, 'Experiments with proof plans for induction', *Journal of Automated Reasoning*, **7**, 303–324, (1991).

[9] R.E. Fikes and N.J. Nilsson, 'STRIPS: A new approach to the application of theorem proving to problem solving', *Artificial Intelligence*, **2**, 189–208, (1971).

[10] M. Gordon, R. Milner, and C.P. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Springer, 1979.

[11] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.

[12] L.M. Hines, 'Str+ve⊂: The stri+ve-based subset prover', in *CADE-10*, (1990).

[13] J. Jaffar, S. Michaylow, P. Stuckey, and R. Yap, 'The CLP(R) language and system', *ACM Transactions on Programming Languages*, **14**(3), 339–395, (1992).

[14] K.R. Koedinger and J.R. Anderson, 'Abstract planning and perceptual chunks: Elements of expertise in geometry', *Cognitive Science*, **14**, 511–550, (1990).

[15] W. McCune, 'Solution of the Robbins problem', *Journal of Automated Reasoning*, **19**(3), 263–276, (1997).

[16] W.W. McCune, 'Otter 2.0 users guide', Technical Report ANL-90/9, Argonne National Laboratory, (1990).

[17] E. Melis, 'A model of analogy-driven proof-plan construction', in *IJCAI-95*, pp. 182–189, Montreal, (1995).

[18] E. Melis, 'Progress in proof planning: Planning limit theorems automatically', Technical Report SR-97-08, (1997). available from http://www.ags.uni-sb.de/publications/deduktion/seki/SR-97/index.html.

[19] S. Minton, C. Knoblock, D. Koukka, Y. Gil, R. Joseph, and J. Carbonell, *PRODIGY 2.0: The Manual and Tutorial*, 1989. CMU-CS-89-146.

[20] F. Stolzenburg, 'Membership constraints and complexity in logic programming with sets', in *Frontiers in Combining Systems*, eds., F. Baader and U. Schulz, 285–302, Kluwer, (1996).

[21] Y. Tetsuya, A. Bundy, I. Green, T. Walsh, and D. Basin, 'Coloured rippling: An extension of a theorem proving heuristic', in *ECAI-94*, pp. 85–89, (1994).