A multi-modi Proof Planner

Lassaad Cheikhrouhou
lassaad@cs.uni-sb.de

Fachbereich Informatik, Universität des Saarlandes
D-66123 Saarbrücken, Germany

Abstract. Proof planning is a novel knowledge-based approach for proof construction, which supports the incorporation of mathematical knowledge and the common mathematical proof techniques of a particular mathematical field. This paradigm is adopted in the ΩMEGA proof development system, to provide support for the user. A considerable part of the proof construction and even sometimes the whole work can be undertaken by a proof planner.
In the ΩMEGA project we are investigating the aspect of computation under bounded resources in mathematical theorem proving. The relevant resources are, in addition to time and memory space, user availability as well as the frequency of user interaction. At this issue, the proof planner of ΩMEGA is conceived in such a way that it has a resource-adaptive behaviour. This property of the planner is achieved by a planner modus which defines the planner behaviour depending on which and how many resources are available. In this paper, we describe the different modi of the ΩMEGA planner and highlight the main aspects of resource-adaptivity.

1 Introduction

Proof planning, as introduced by A. Bundy [Bun91], is a knowledge-based approach for proof construction. It is based on the search for a sequence of proof operators that constitutes a proof plan which in turn can then be applied to construct a proof on the basic calculus (object) level. The proof operators, called methods, are specifications of tactics, where a tactic consists of a coherent composition of many basic inference rules. A method states in a meta-language when a tactic can be applied and what its effects may be.

The paradigm of proof planning is adopted in the ΩMEGA system [BCF+97], which is an interactive environment for proof development in the domain of mathematics, to provide a support for the user. Besides calling tactics or deliberating subproblems to one of the integrated automatic theorem prover, a proof planner can be invoked which is able to undertake the whole or a considerable part of the proof work.

According to the theoretical properties of the search for a proof, a sensible result cannot be always expected from an automatic proof planner, especially in the domain of mathematics. In order to allow user intervention, we developed a plan-based proof planner which can be called in different modi. Besides using the planner as a black-box in the *automatic modus*, the user can intervene in the search control of the planner both in the *interactive modus* and in the *mixed-initiative modus* to provide some guidance for the search process. In contrast to the ΩMEGA original planner, a state-based planner [HKRS94], the explicit representation of the partial plan in the new planner enables the user to pursue the progress of the proof plan construction.

In the ΩMEGA project the aspect of computation under bounded resources in mathematical theorem proving is being investigated [SKM]. The relevant resources are, in addition to time and memory space, user availability as well as the frequency of user interaction. At this issue, the proof planner of ΩMEGA is conceived in such a way that it has a resource-adaptive behaviour. Following the terminology in [Zil95], the planner has to act with bounded rationality by adaptation. This property of the planner is achieved by a planner modus which defines the planner behaviour depending on which and how many resources are available. Each of the modi mentioned above possesses some parameters for resource allocation.

In this paper, we present a multi-modi, resource-adaptive proof planner which is implemented and integrated in the ΩMEGA system. We used this planner to construct several proofs, for instance, a proof plan for the Cantor theorem is computed automatically with the help of the diagonalization method [Che97]. Before describing the different modi of this planner and highlighting the main aspects of resource-adaptivity, we give an overview of the data structure used to represent partial proof plans and of the operators the planner can employ.

2 Proof Plan Data Structure

The entire process of theorem proving in ΩMEGA can be viewed as an interleaving process of proof planning, plan execution, and verification that is centered around the *Proof Plan Data Structure* ($\mathcal{PDS}$), shown in Fig. 1.

The hierarchical data structure represents a partial proof of the theorem to be shown at several levels of abstraction. The levels correspond to different abstractions of the formal proof in the calculus of natural deduction (ND) [Gen35]. A $\mathcal{PDS}$ level is a proof tree with the theorem as its root and the hypotheses as its leaves. The child nodes in such a proof tree are either the premises for the justification of their *closed* parent node or the *supports* for their *open* parent node. The supports of an *open* node consist of a heuristically determined subset of the hypotheses of this node and of their deduced consequences, which should be used to close this open node.

A justification of a closed node denotes that the conjecture of this node follows logically from those of the premises by a named *inference*. A *base inference* is just one application of a single ND rule, whereas a *non-base inference* can be a tactic, a proof schema that represents a proof idea (technique) with still some gaps, or a call of an external reasoner, such as an automatic theorem prover, for instance, `OTTER` [McC90], or a computer algebra system. The returned result of an external reasoner must be transformed into an ND proof. A non-base infer-

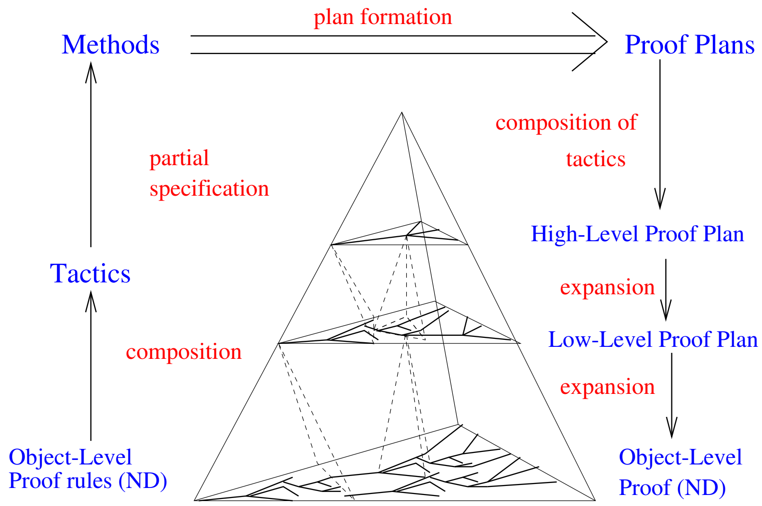

Fig. 1. Proof Plan Data Structure

ence abstracts many individual inference steps by grouping them into one proof step at a higher level of abstraction.

An initial $\mathcal{PDS}$ consists of an open node, the theorem, and its supports, the hypotheses. A formal ND proof for this problem is constructed by the application of inferences to open nodes and the expansion of non-base inferences. This construction process can be carried out interactively and/or with the help of a planner which uses *methods* as plan operators. A *method* is a specification that represents the applicability condition and application effects of an inference (see section 3).

We call the data structure for representing proofs a *proof plan data structure*, because we interpret this as a proof plan at different levels (similar to the hierarchical representation of plans in for example `SIPE` [Wil88]). The methods correspond to the operators, where the methods that specify the application of base inferences, i.e. ND rules, are seen as primitive actions. The expansion of a $\mathcal{PDS}$ level to a less abstract level can be considered as the refinement of a plan.

The conjectures in the $\mathcal{PDS}$ can contain partially specified terms, i.e. terms with meta-variables. These meta-variables are place holders for object terms that are partially specified by some constraints that become more and more refined as the search for a proof plan continues. Contrary to AI-planning, there is principally no interaction between fully instantiated goals in proof planning. However, partially instantiated goals can interact through common meta-variables or meta-variables related by common constraints. Therefore, the meta-variables constraints are managed in a constraint store parallel to the $\mathcal{PDS}$.

3 Methods

Methods can be seen as operators that manipulate a partial $\mathcal{PDS}$ with the aim of obtaining a complete $\mathcal{PDS}$, i.e., a proof plan that is expandable into a formal ND proof. Each method corresponds to some inference, it specifies when the associated inference can be applied and what its effects may be.

In ΩMEGA, we use a declarative representation as much as possible for methods, such that meta-methods can be employed to reformulate methods in analogy-driven theorem proving [Mel98]. A method is represented as a frame-like structure that consists of the following slots (see Fig. 2 and Fig. 3 for examples). Sorted variables, called *method variables*, are used as place-holder for $\mathcal{PDS}$ objects. The sort of a method variable corresponds to the syntactic designation of the $\mathcal{PDS}$ object represented by this variable.

Method : `EquivIm`				
Conc..s	($\ominus$ 4)			
Prem..s	($\oplus$ 1) ($\oplus$ 2)			
Proof Schema	1.	$\vdash$	$\phi \to \psi$	`(Open)`
	2.	$\vdash$	$\psi \to \phi$	`(Open)`
	3.	$\vdash$	$(\phi \to \psi) \wedge (\psi \to \phi)$	`(AndI 1 2)`
	4.	$\vdash$	$\phi \equiv \psi$	`(DefnI 3)`

Fig. 2. An example of a backward Method

- The *proof schema* represents the proof part that should be inserted into a $\mathcal{PDS}$ when expanding the method. It consists of *method nodes* that are matched to ND lines, and of *method variables* which are place holder for lists of ND lines.
  Similar to an ND line, a *method node* consists of a label, of a schematic formula with possibly free method variables, of a hypothesis list containing the extra hypotheses that are newly inserted by this method, and of a justification. A justification of a method node can be either open (annotated by `Open`) or closed. A closed justification consists of an inference, e.g., an ND rule, an argument list, and a list of method nodes (the premises). This means, the corresponding method node can be proven from the premises by applying the inference with the given arguments.
  Generally, a method variable specifying lists of ND lines refers to a proof part which has to be generated by a procedural call, e.g., by an automatic theorem prover, and which cannot be represented schematically with the help of method nodes.
- The *conclusions* comprise annotated method nodes that correspond to the ND lines closed by the method. Generally, the conclusion list of a *backward*

Method : `AndEm`				
Conc..s	$(\oplus 3)$ $(\oplus 2)$			
Prem..s	$(\ominus 1)$			
Proof Schema	1.	$\vdash$	$\phi \wedge \psi$	()
	2.	$\vdash$	ψ	`(AndER 1)`
	3.	$\vdash$	ϕ	`(AndEL 1)`

Fig. 3. An example of a forward Method

method, for instance `EquivIm` in Fig. 2, consists of a method node annotated with $\ominus$, that should match an open node in the $\mathcal{PDS}$ which is reduced by the method application to new open nodes. In contrast, the conclusion list of a *forward* method, for instance `AndEm` in Fig. 3, contains only method nodes annotated with $\oplus$. The ND lines associated to these conclusions are added into the $\mathcal{PDS}$ as new closed nodes justified by the method.

- The *premises* consist of possibly annotated method nodes that correpond to the ND lines, from which the conclusions logically follow. The associated ND line to a premise annotated by $\oplus$ is added as an open node to the $\mathcal{PDS}$ and can be regarded as a subgoal of the method. This new open node inherits the supports of the goal, the method contributes to its proof. The non-annotated premises and the premises annotated by $\ominus$ must match some supports of the considered goal.
  The premises and the conclusions are called the *outlines* of the method, because they correspond to the boundaries of the method proof which are significant for the applicability of the method. The outlines annotated with $\oplus$ are denoted by *add-outlines* and the rest are designated by *exist-outlines*.
- The *application condition* specifies some additional restrictions for the application of the method. These conditions are evaluated after successfully matching the exist-outlines to appropriate ND lines. They permit to represent additional application requirements which cannot be checked by the matching procedure, which is a first order alpha-matcher. Such conditions are not needed in methods where the application requirements can be represented by the formula-schemas of the exist-outlines (compare the `AndEm` method in Fig. 3).

Other slots, which are not needed for each method, enable to prescribe additional control and computation information. For instance, the *application constraint* restricts the binding domain of meta-variables which are considered or inserted by the method, its evaluation changes therefore, the constraint store of the $\mathcal{PDS}$. The *application computations* and the *expansion computations* correspond to some function calls which compute parts of the proof tree that cannot be represented declaratively. It is possible to represent a partial order on the method subgoals in the *orderings* slot, and to instruct how to change the supports of some method subgoals by the *application actions*.

We distinguish *reliable* and *non-reliable* methods. A method is non-reliable, when its expansion can lead to an incomplete subproof, i.e., with open nodes. For instance, a method that calls a resolution theorem prover, e.g., OTTER, is non-reliable, because the considered problem may not be solved by this prover within a given resource limit. The expansion of a reliable method on the other hand, e.g., a specification of a tactic application, always leads to a complete subproof at a more detailed level. This distinction is important, because, during the search for a proof, only open nodes and nodes justified by non-reliable methods are taken into consideration by the planner.

4 Planning Algorithm

While constructing a $\mathcal{PDS}$ we focus on the current tasks by keeping them in an agenda, which is a set of partially ordered tasks. We distinguish three types of tasks: *goals*, that correspond to the current open nodes with fully instantiated formulas, *goal-schemas*, which consist of the current open nodes whose formulas contain meta-variables, and *pseudo-goals*, which are the nodes justified by, not yet expanded, non-reliable methods. The ordering constraints between some tasks are inserted by the methods generating these tasks.

Initially, the agenda contains the theorem as the sole goal. Constructing a proof by the planner amounts to solving the tasks on the agenda in a systematic way. Hereby, we use a hierarchical planner that searches in the space of partial plans. In contrast to typical hierarchical planners like SIPE [Wil88], which generally apply operators to close the goals on some planning level and then expand the whole plan to consider the new goals on the next planning level, our planner interleaves method application to close the goals and expansion of non-reliable methods to possibly insert new goals on a less abstract level. Consequently, the search for a proof plan by the ΩMEGA planner is done in one planning level. A planning step, as described in Fig. 4, consists of either applying a method, or expanding a non-reliable method, when this succeeds, and corresponds otherwise to a backtracking step (see instructions 5 and 6 in Fig. 4).

The first instruction of the planning step procedure described in Fig. 4 retrieves the first tasks on the $\mathcal{PDS}$ agenda. According to a control principle which delays the treatment of goal-schemas with the objective of getting maximal information to bind the involved meta-variables, goals and pseudo-goals are considered first (see instruction 2 in Fig. 4). A schematic goal can be considered in the planning step, only if no goal and no pseudo-goal exists in the agenda first tasks.

Next, a task is determined to focus on in the rest of the planning step (see instructions 3 and 4 in Fig. 4). When there is only one goal or pseudo-goal to be considered, then this would be the task for the sequel of the planning step. Otherwise, there can be good reasons to prefer, select, or reject some tasks with the help of appropriate control rules which are defined wrt. to the planning situation. For instance, some kinds of interactions between schematic goals caused by related meta-variables can select one of the interacting goals. Provided con-

```
PlanStep(pds):
1. tasks ← AgendaFirstTasks(pds)
2. tasks′ ← tasks \ GoalSchemata(tasks)
3. if Empty(tasks′)
     then task-and-methods ← TaskAndMethods(tasks,pds,MOR-Methods)
     elsif Rest(tasks′)
          then task-and-methods ← TaskAndMethods(tasks′,pds,Methods)
          else task-and-methods.Task ← First(tasks′)
4. task ← Task(task-and-methods)
5. if PseudoTask(task)
     then success ← RefineTask(task,pds)
     else success ← ApplyFirstMethod(task-and-methods,pds)
6. when Not(success)
     Backtrack(task,pds)
```

Fig. 4. A sketch of the planning step procedure

trol rules are available for this purpose, they are evaluated by a control rule interpreter (CRI) which appoints one task from the alternative tasks with possibly some additional control information on how to achieve this task, e.g., some preferred methods and some preferred support lines. The CRI is invoked even when the agenda first tasks consist of only one schematic goal, while such a task can interact via meta-variables with some of the remaining tasks on the agenda.

The final iteration in the planning step is trying to accomplish the selected task using possibly additional control information suggested by the CRI. When the selected task corresponds to a pseudo-goal, then the justifying method is expanded by introducing the associated proof schema in the $\mathcal{PDS}$. The considered task is replaced on the agenda by new tasks associated to new open nodes and involved nodes justified by non-reliable methods.

When the selected task corresponds to a goal or to a goal-schema, then the first applicable *relevant* method is used to close this goal. According to the type of the task at hand, the relevant methods belong either to MOR-Methods or to Methods and were not considered in a previous planning step to achieve the same task. The elements of Methods are employed to close fully instantiated goals. Those of MOR-Methods implement generally heuristics and special techniques for Middle-Out-Reasoning (MOR) [KBB93] and are designated to handle schematic goals.

The methods in MOR-Methods and in Methods are sorted according to a preference value, the *method rating*. The assignments of method ratings predefined the default search strategy of the planner, for instance, in order to be considered first, backward methods are associated higher rating values than forward methods. The default order of a task relevant methods can be changed by a CRI call.

Considering all relevant methods, first the methods sorted by the CRI, the foremost applicable method is used to contribute to the solving of the selected task. While applying this method new tasks can be inserted in the agenda relative to the considered task. This task is removed from the agenda, when the applied method is a backward method reducing the corresponded goal to new subgoals. A partial order of these new tasks can be specified by the method orderings. Moreover, the remaining not yet considered relevant methods are stored in the task associated node as other alternatives for possible backtracking.

When the expansion of a method fails or no relevant method is applicable to the considered goal, then the planner has to backtrack. We adopt a hybrid backtracking approach, chronological wrt. the modification of the constraint store and dependency-directed wrt. the subgoaling relation in the $\mathcal{PDS}$. When a failed goal-schema is introduced in a planning step which preceeds the planning step that last changed the constraint store, then the latter planning step is taken back at first. The constraint store is reset to an earlier state and the failed task can be considered again. In other cases, including the considered task corresponds to a goal without meta-variables, a dependency directed backtracking is followed and the plan step that introduced this task is taken back.

Similar to a tactic, the procedure described in Fig. 4 can be interactively called by the user to perform one planning step. More interesting, is to invoke the planner in order to construct a proof plan for the problem at hand automatically. The straightforward realization of a planning process is to call the planning step procedure in a loop until the $\mathcal{PDS}$ agenda is empty. With such a conventional planner, there is no opportunity for the user to intervene in the search process only by interrupting the search loop. In the next section, we describe the different modi of a resource-adaptive planner, where the user can influence the progress of the planning process. The relevant resources are in addition to time and memory space, user availability as well as the frequency of user interaction.

5 The Planner Modi

In the ΩMEGA system, the aspect of resources in mathematical theorem proving is being investigated [SKM]. At this issue, the proof planner is conceived in such a way that it has a resource-adaptive behaviour. This property of the planner is achieved by a planner modus which defines the planner behaviour depending on which and how many resources are available.

5.1 Automatic Modus

In the *automatic* modus, the planner does not expect any user support. With the help of an explicit bound, the maximal number of planning steps, it is possible to assign the only available resources time and place, provided the number of steps is proportional to the search time and the needed space. The other modi, described below, naturally depend on the same bound too.

Principally, the planner in the automatic modus repeats the call of the planning step procedure in Fig. 4 until a complete $\mathcal{PDS}$ is constructed, i.e., the $\mathcal{PDS}$ agenda is empty, or the maximal number of planning steps is exhausted. When the bound of maximal planning steps is ∞ and no complete $\mathcal{PDS}$ can be found, the user is confronted with the problem of an infinite loop and the only possible user intervention is to break the search process. The next modus offers more elaborated opportunities to repare such planning situation.

5.2 Mixed-initiative Modus

In a mixed-initiative planner, the search control depends on the planners own decisions as well as on the user decisions. This modus assumes the availability of a user who, through the graphical user interface $\mathcal{L}\Omega\mathcal{UI}$ [SHB+98], follows the progress of the $\mathcal{PDS}$ construction process and can propose decisions to influence this process at any given status.

The current state of the $\mathcal{PDS}$ is displayed as a tree together with the agenda. $\mathcal{L}\Omega\mathcal{UI}$ provides more visualization options of the $\mathcal{PDS}$ content. In order for the user to be able to assimilate the progress of the planning process, the frequency of displaying the $\mathcal{PDS}$ is diminished by a delay which can be adapted to the user capability. The user can intervene in the construction process of the $\mathcal{PDS}$ by simultenously positioning the mouse either on an agenda task or on a $\mathcal{PDS}$ node and selecting one of the possible decisions displayed in a pop-up-menu. The suggested user decision is then sent by $\mathcal{L}\Omega\mathcal{UI}$ to the planner which has to perform it. Before deciding to carry out a planning step as described in Fig. 4, the planner inspects a communication port for messages from $\mathcal{L}\Omega\mathcal{UI}$ reserved for user decisions to execute such decisions first. Some of possible user decisions are listed below:

- Deleting a whole subtree of the $\mathcal{PDS}$ in case there is a possible infinite loop.
- Taking back an applied method by opening its conclusions.
- Delaying the handling of some subgoals, for instance, when a very deep $\mathcal{PDS}$ path ends in some subgoal and the consideration of this subgoal could lead to an infinite loop. At the same time, the user is interested in the rest of the proof, then it is possible to block the deep task as if it were closed and then pursue whether the planner can achieve the remaining tasks.
- Selecting some important task to be considered first and possibly proposing a list of methods to be preferred.
- Changing the planner modus, for instance, when the user estimates all the remaining tasks to be obvious for the planner, he can change to the automatic modus.

5.3 Interactive Modus

In the interactive modus, it is possible to state in addition to the bound of maximal planning step, two other bounds which specify how often the planner may disturb (bother) the user. These parameters are maximal number of alternatives for each conflict situation, where one of the possible alternatives has to be chosen in the planning process. Up to these maximal numbers, the planner has to make the choice itself. Whenever the number of alternatives in a conflict situation exceeds the associated bound, the planner breaks the search and deliberates the decision to the user.

The procedure of an interactive planning step corresponds to a little modified version of the procedure described in Fig. 4. The routines TASKANDMETHODS and APPLYFIRSTMETHOD are respectively replaced by USERTASK and APPLYUSERMETHOD. In the resulted procedure, we distinguish two conflict situations, each one is associated a maximal number.

The *maximal tasks number* is designated to the choice of a task from the first tasks on the agenda. This is taken into account when calling the routine USERTASK which invokes the CRI if the number of first agenda tasks preceeds the bound of *maximal tasks* and otherwise asks the user to select one of these tasks. A second bound, the *maximal methods* is assigned to the selection of one of the applicable methods. The routine APPLYUSERMETHOD checks the applicability of all relevant methods. When the number of applicable methods exceeds the bound of *maximal methods*, then the user is asked to select one of them, otherwise this routine decides, eventually with the help of the CRI, which of these methods is to be applied first.

Setting both bounds to 1 corresponds to a fully interactive planner, where each control decision must be taken by the user. The other extreme, assigning ∞ to both bounds, results in the fully automatic planner described in section 5.1. We speculate that a function which delivers the appropriate values of these bounds given the target rate of interactions can be statistically determined. Hereby, we can somehow control the assignment of the resource user interaction.

6 Conclusion and Related Works

We presented a multi-modi, resource-adaptive proof planner which is implemented and integrated in the ΩMEGA system. We used this planner to construct several proofs, for instance, a proof plan for the Cantor theorem is computed automatically with the help of the diagonalization method [Che97]. The principle ideas and functionalities of the interactive modus and of the mixed-initiative modus could be shown by the prototype implementation. We still work on the improvement of the user-system communication to reach a comfortable use of these modi. We think about more elaborated techniques to provide a flexible and user-adapted abstract view of the progress in the search process.

The Edinburgh group of A. Bundy is the outrider of adopting the AI planning approach in the search of theorem proofs. Its CLAM system [BvHHS90], originally used for proofs by induction, is a state-based forward planner (see [BvHHS91], page 7 and 8) with many available search strategies, which are seperately used.

Cox and Veloso combined a framework for interactive planning ForMAT and an automatic planner PRODIGY/ANALOGY to a mixed-initiative planner. In [CV97], they adapted the control of the automatic planner by developing special control rules which better interpret the partial plans constructed by the user to reach a major contribution by the automatic planner to solving the planning problem.

References

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a Mathematical Assistant. In W. McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, LNAI, Townsville, Australia, 1997. Springer Verlag.

[Bun91] A. Bundy. A Science of Reasoning. In *Computational Logic: Essays in honor of Alan Robinson*. MIT Press, 1991. also presented at the 10th CADE 1990 as extended abstract.

[BvHHS90] A. Bundy, F. van Harmelen, Ch. Horn, and A. Smaill. The OYSTER-CLAM system. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 647–648, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[BvHHS91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, **7**:303–324, 1991. Also pubished as DAI Research Paper No. 413.

[Che97] L. Cheikhrouhou. Planning Diagonalization Proofs. In G. Brewka, Ch. Habel, and B. Nebel, editors, *Proceedings of KI-97, 21st German Annual Conference on Artificial Intelligence*, pages 377–380, Freiburg, Germany, 1997. Springer Verlag, Berlin, Germany, LNAI 1303.

[CV97] M. T. Cox and M. M. Veloso. Controlling for Unexpected Goals when Planning in a Mixed-Initiative Setting. In E. Costa and A. Cardoso, editors, *Progress in AI: 8-th Portuguese Conference on AI, EPIA97*, Coimbra, Portugal, October 1997. Springer Verlag, Berlin, Germany, LNAI 1323.

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, **39**:176–210, 1935.

[HKRS94] X. Huang, M. Kerber, J. Richts, and A. Sehn. Planning Mathematical Proofs with Methods. *Journal of Information Processing and Cybernetics, EIK*, **30**(5-6):277–291, 1994.

[KBB93] I. Kraan, D. Basin, and A. Bundy. Middle-Out Reasoning for Program Synthesis. In P. Szeredi, editor, *Proceedings of the 10-th International Conference on Logic Programming*. MIT Press, 1993.

[McC90] W. McCune. Otter 2.0. In M. E. Stickel, editor, *Proceedings of the 10th CADE*, pages 663–664, Kaiserslautern, Germany, 1990. Springer Verlag, Berlin, Germany, LNAI 449.

[Mel98] E. Melis. The Heine-Borel challenge problem: In honor of Woody Bledsoe. *Journal of Automated Reasoning*, 21, 1998. forthcoming.

[SHB+98] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, E. Melis, A. Meier, and V. Sorge. LΩUI: A Distributed Graphical User Interface for the Interactive Proof System ΩMEGA. 1998. forthcoming.

[SKM] J. H. Siekmann, M. Kohlhase, and E. Melis. ΩMEGA: Ein mathematisches Assistenzsystem. *Kognitionswissenschaft*. forthcoming.

[Wil88] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., Artificial Intelligence Center SRI International, 1988.

[Zil95] Sh. Zilberstein. Models of bounded rationality. In *AAAI Fall Symposium on Rational Agency*, Cambridge, Massachusetts, November 1995.