ΩMEGA: Resource Adaptive Processes in an Automated Reasoning System

Serge Autexier, Christoph Benzmüller, Dominik Dietrich, Jörg Siekmann

Abstract This paper summarises and gives an overview of the development of the ΩMEGA system during the 12 years of funding by the SFB 378.

1 Motivation

The research objective of the ΩMEGA project has been to lay the foundation complex, heterogenous, but well integrated assistance systems for mathematics, which support the wide range of typical research, publication and knowledge management activities of a working mathematician. Examples are computing (for instance algebraic and numeric problems), proving (lemmas or theorems) , solving (for instance equations), modelling (by axiomatic definitions), verifying (typically a proof), structuring (for instance the new theory and knowledge base), maintaining (the knowledge base), searching (in a very large mathematical knowledge base), inventing (your new theorems), paper writing, explaining and illustrating in natural language and diagrams. Clearly, some of them require a high amount of human ingenuity

Serge Autexier
DFKI GmbH and Saarland University, Campus, Geb. E11, 66123 Saarbrücken, Germany e-mail: autexier@dfki.de

Christoph Benzmüller
FR 6.2 Informatik, Saarland University, Campus, Geb. E11, 66123 Saarbrücken, Germany e-mail: chris@ags.uni-sb.de

Dominik Dietrich
FR 6.2 Informatik, Saarland University, Campus, Geb. E11, 66123 Saarbrücken, Germany e-mail: dietrich@ags.uni-sb.de

Jörg Siekmann
DFKI GmbH and Saarland University, Campus, Geb. E11, 66123 Saarbrücken, Germany e-mail: siekmann@dfki.de

while others do not and they are thus open to computer support with current AI and Computer Science technology.

Our research is based in particular on the combination of techniques from several subfields of AI including knowledge representation and reasoning, cognitive architectures and multi-agent systems, human computer interaction and user interfaces, as well as machine learning, intelligent tutor systems and finally dialog systems with natural language processing capabilities.

The central notion for the integration of these techniques is that of a *resource* and the adaptation of the system to a wide range of resources. In fact, a mathematical assistant system can be considered as a performance enhancing *resource for human users* as well as a *resource for other software systems* using it. Furthermore, a user friendly mathematical assistance system has to solve a given task within *limited time and space resources*. While executing a task, let's say, automatically planning a proof for a theorem, the systems performance may significantly depend on further *knowledge resources* such as proof methods, theorems and lemmas. Furthermore, the assistant system may exploit *specialised computing and reasoning resources*, for example, an external computer algebra system, an automated deduction system or a model generator.

Considering a mathematical assistance system itself as a resource requires the development of different interfaces—for a human user or for other software systems. This in turn poses the problem how the system adapts itself to such conceptually very different means of interaction.

The article is organised as follows: Section 2 presents proof representation and proof search techniques that utilise knowledge and specialised computing resources. We discuss the representation, authoring, access to and maintenance of knowledge resources in Section 3 and specialised computing resources in Section 4. Section 5 develops the infrastructures and internal architecture that enables the assistance system to adapt to the different means of interaction.

2 Resource-Adaptive Proof Search

This section presents the proof search techniques that exploit different resources to prove a given conjecture. The proof procedures all work on a central, elaborate *proof object*, which supports the simultaneous representation of the proof at different levels of granularity and records also alternative proof attempts.

2.1 Human-oriented high-level Proofs

The central component of our computer-based proof construction in ΩMEGA is the TASKLAYER. It is based on the CORE-calculus (Autexier, 2005) that supports proof development directly at the *assertion level* (Huang, 1996), where proof steps are justified not only by basic logic inference rules but also by definitions, axioms, theorems or hypotheses (collectively called *assertions*).

Subgoals to be shown are stored within the TASKLAYER as *tasks*, which are represented as Gentzen-style multi-conclusion sequents (Gentzen, 1969). In addition there are means to define multiple foci of attention on subformulas that are maintained within the actual proof. Each task is reduced to a possibly empty set of subtasks by one of the following proof construction steps: (1) the introduction of a proof sketch (Wiedijk, 2004), (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be postulated on the fly (and proved later), (4) the substitution into meta-variables, and (5) the application of an *inference*. Inferences are the basic reasoning steps of the TASKLAYER, and comprise assertion applications, proof planning methods or calls to external special systems such as a computer-algebra system, an automated deduction system or a numerical calculation package (see (Dietrich, 2006; Autexier and Dietrich, 2006) for more details about the TASKLAYER).

2.1.1 Inferences

Intuitively, an *inference* is a proof step with multiple premises and conclusions augmented by (1) a possibly empty set of hypotheses for each premise, (2) a set of *application conditions* that must be fulfilled upon inference application, (3) a set of *completion functions*[1] that compute the values of premises and conclusions from values of other premises and conclusions, and (4) an *expansion function* that refines the abstract inference step. Each premise and conclusion consists of a unique name and a formula scheme. Note that we employ the term *inference* in its more general psychological meaning: taken in that sense, an inference may turn out to be invalid actually, in contrast to the formal logical notion of an *inference rule*.

Additional information needed in the application conditions or the completion functions, such as, for instance, the position of a subterm or the instance of some non-boolean meta-variable, can be specified by additional *parameters* to the inference.

Application conditions are predicates on the values of inference variables and completion functions compute values for specific inference variables from values of other inference variables.
An example of an inference is given in Fig. 1. The inference *subst-m* has two premises p_1,p_2 with formula schemes F and $U = V$ respectively, one conclusion c with formula scheme G, and one parameter π. It represents the inference that if we are given a formula F with subterm U at position π and the equation $U = V$, then we can infer the formula G which equals F except that U is replaced by V. The completion functions are used to compute the concrete conclusion formula c, given p_1, p_2, and π. They can also be used for the "backward" direction of the inference to compute the formula p_1, given c, p_2, and π, or to compute the position π at which a

[1] The completion functions replace the "outline functions" in previous work.

$$\frac{p_1 : F \quad p_2 : U = V}{c : G}\,\textit{subst-m}(\pi)$$

Appl. Cond.: $(F|_\pi = U \wedge G|_{\pi \leftarrow V} = F) \vee (G|_\pi = U \wedge F|_{\pi \leftarrow V} = G)$
Completions: $\langle c, \texttt{compute-subst-m}(p_1, p_2, \pi)\rangle$
$\langle p_1, \texttt{compute-subst-m}(p_2, c, \pi)\rangle$
$\langle \pi, \texttt{compute-pos}\ (p_1, p_2)\rangle$
$\langle \pi, \texttt{compute-pos}\ (p_2, c)\rangle$

Fig. 1: Inference *subst-m*

replacement can be done. Note that there are two completion functions for computing π. Furthermore, for a given formula F for p_1 and equation $U = V$ for p_2, there are in general more than one possible value for π. Therefore, the completion functions actually compute *streams* of values, each value giving rise to a new instance of the inference.

Inferences can also encode the operational behaviour of domain specific assertions. Consider for instance the domain of set theory and the definition of $\subseteq$:

$$\forall U, V. U \subseteq V \Leftrightarrow (\forall x. x \in U \Rightarrow x \in V)$$

That assertion gives rise to two inferences:

$$\frac{\begin{array}{c}[x \in U]\\ \vdots \\ p : x \in V\end{array}}{c : U \subseteq V}\,\textit{Def-}\subseteq \qquad \frac{p_1 : U \subseteq V \qquad p_2 : x \in U}{c : x \in V}\,\textit{Def-}\subseteq$$

Appl. Cond.: x *new for* U *and* V **Appl. Cond.:** x *new for* U *and* V

As a result, we obtain proofs where each inference step is justified by a mathematical fact, such as a definition, a theorem or a lemma.

To illustrate the difference between a typical proof step from a textbook and its formal counterpart in natural deduction consider the assertion step that derives $a_1 \in V_1$ from $U_1 \subset V_1$ and $a_1 \in U_1$. The corresponding natural deduction proof is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall U, V.\ U \subset V \Leftrightarrow \forall x. x \in U \Rightarrow x \in V}{\forall V.\ U_1 \subset V \Leftrightarrow \forall x \in U_1 \Rightarrow x \in V}\,\forall_E}{U_1 \subset V_1 \Leftrightarrow \forall x. x \in U_1 \Rightarrow x \in V_1}\,\forall_E}{U_1 \subset V_1 \Rightarrow \forall x. x \in U_1 \Rightarrow x \in V_1}\,\Leftrightarrow_E \quad U_1 \subset V_1}{\forall x. x \in U_1 \Rightarrow x \in V_1}\,\Rightarrow_E}{a_1 \in U_1 \Rightarrow a_1 \in V_1}\,\forall_E \quad a_1 \in U_1}{a_1 \in V_1}\,\Rightarrow_E$$

Even though natural deduction proofs are far better readable than proofs in machine oriented formalisms such as resolution, we see that they are at a level of detail we hardly find in a proof of a typical mathematical textbook or in a research publication: In the example above, a single assertion step corresponds to 6 steps in the natural deduction calculus.

Similarly, the lemma $\forall U, V, W.(U \subseteq V \wedge V \subseteq W) \Rightarrow U \subseteq W$ stating the transitivity of $\subseteq$ can be represented by the inference:

$$\frac{p_1 : U \subseteq V \qquad p_2 : V \subseteq W}{c : U \subseteq W}\,\textit{Trans-}\subseteq \tag{1}$$

An eminent feature of the TASKLAYER is that inferences can be applied to *sub*-formulas of a given task. Consider the task to prove that if f is an automorphism on some group G, then the f-image of the Kernel of G is a subset of G.

$$A \subseteq B \Rightarrow f(A) \subseteq f(B) \vdash \textit{AutoMorphism}(f, G) \Rightarrow f(\textit{Ker}(f, G)) \subseteq G \tag{2}$$

where we have the additional hypothesis, that if two arbitrary sets are in a subset relation, then so are their images under f.

A deep application of the inference *Trans-* $\subseteq$ matching the conclusion with the subformula $f(\textit{Ker}(f, G)) \subseteq G$ and the first premise with the subformula $f(A) \subseteq f(B)$ reduces the task in one step to

$$A \subseteq B \Rightarrow (f(A) \subseteq f(B)) \vdash \textit{AutoMorphism}(f, G) \Rightarrow (\textit{Ker}(f, G) \subseteq B \wedge f(B) \subseteq G)$$

which can be proved immediately using the definitions of *AutoMorphism* and *Ker*. This one step inference would not be possible unless we can use (1) to match the subformula within the conclusion of the task (2).

2.1.2 Application Direction of an Inference

The problem is to find all possible ways to apply a given inference to some task, i.e. to compute all possible instantiations of an inference. Typically, some of the parameters as well as some of the formal arguments of the inference are already instantiated. The formal arguments and the parameters of an inference will be collectively called the *arguments* of an inference.

The process starts with a partial argument instantiation (PAI) and we have to find values for the non-instantiated arguments of the inference. These arguments take positions as values within the task or they have formulas as values.

Example 1. Consider the inference *subst-m* of Fig. 1 before, which we want to apply to the task T: $2*3=6 \vdash 2*3<7$. Then $\textit{pai}_1 = \langle \emptyset, \{c \mapsto (10)\}, \emptyset \rangle$ is a partial argument instantiation for the inference *subst-m*, where (10) denotes the position of $2*3<7$ in the task. As no completion function has been invoked so far, $\textit{pai}_1$ is *initial*. It is not *complete* as there is not enough information for the completion functions to compute π given only c; thus $\mathsf{p}_1, \mathsf{p}_2$ can not be computed yet.

The extension of a partial argument instantiation consists of an assignment of values to arguments of the inference that are not yet instantiated. There are two possible choices: (1) either assign a task position to a formal argument or (2) to assign a term to a formal argument.

The first kind of update involves searching for possible positions in the task while respecting already introduced bindings. The second kind of update involves no search for further instances in the task, as it uses the completion functions to compute the missing values.

Thus we can separate the updating process into two phases: In the first phase we update the positions by searching the task for appropriate instance of the formula schemes and in the second phase we only use the completion functions to compute the missing arguments. The general idea is to use as much derived knowledge as possible and then decide whether this knowledge is sufficient for the inference to be drawn. A partial argument instantiation *pai* is called complete, if it contains sufficient information to compute all other values of arguments using completion functions.

Example 2. If we add an instantiation for the argument p_2 in pai_1 we obtain $pai_2 = \langle \emptyset, \{\mathsf{p}_2 \mapsto (00), \mathsf{c} \mapsto (10)\}, \emptyset \rangle$, where (00) denotes the position of the formula $2 * 3 = 6$ in our task and pai_2 is an extension of pai_1. It is complete, as we can invoke the completion functions to first obtain π and then to obtain p_1.

The configuration of a complete partial argument instantiation describes an *application direction* of the inference. All application directions can be determined by analysing the completion functions of an inference. As an example consider inference *Trans-* $\subseteq$ (p. 5): The application of the inference on task (2) instantiates c with $\mathsf{f}(Ker(\mathsf{f}, \mathsf{G})) \subseteq \mathsf{G}$ and p_1 with $\mathsf{f}(\mathsf{A}) \subseteq \mathsf{f}(\mathsf{B})$, which is represented by the partial argument instantiation $pai := \langle \{\mathsf{p}_1 \mapsto \mathsf{f}(\mathsf{A}) \subseteq \mathsf{f}(\mathsf{B})\}, \emptyset, \{\mathsf{c} \mapsto \mathsf{f}(Ker(\mathsf{f}, \mathsf{G})) \subseteq \mathsf{G}\} \rangle$. The *configuration* of pai_1, that is the premises and conclusions that are instantiated and those which not, classify this rule application as “backward”. The same rule with both premises instantiated but not the conclusion is a “forward” rule.

2.1.3 Representation of Proof

The proof data structure is at the centre of such a system and its task is to maintain the current status of the proof search so far and to represent it at different levels of abstraction and granularity. The *proof data structure* (PDS) is based[2] on the following ideas:

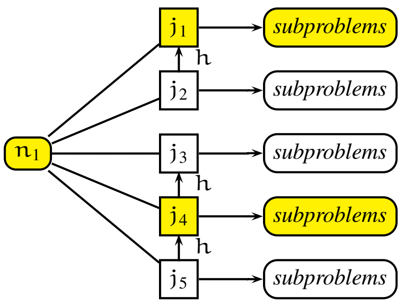
[2] It reflects our experience of more than a decade of development of the ΩMEGA system (Cheikhrouhou and Sorge, 2000; Siekmann et al, 2002a,b, 2003; Autexier et al, 2006) as well as ideas from the QUODLIBET system (Avenhaus et al, 2003).

- Each conjectured *lemma* gets its own *proof tree* (actually a directed acyclic graph), whose nodes are (sub-)goals to be shown.
- In this *proof forest*, each lemma can be applied in each proof tree as an inference; either as a lemma in the usual sense, or as an induction hypothesis in a possibly mutual induction process, see (Wirth, 2004).
- A lemma is a *task* to be proved reductively. A *reduction* step reduces a *goal* to a conjunction of *sub-goals* with respect to a *justification*. This are the proof construction steps of the TASKLAYER.
- Several reduction steps applied to the same goal result in alternative proof attempts, which either represent different proof *ideas* or the same proof idea but at a different level of abstraction or *granularity* (with more or less detail).

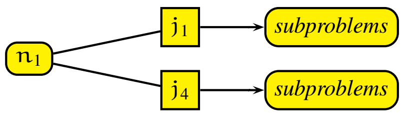
The PDS is essentially a directed acyclic graph (dag) whose nodes are labelled with tasks. It has two sorts of links: *justification hyper-links* represent some relation of a task node to its sub-task nodes, and *hierarchical edges* point from justifications to other justifications which they refine.

This definition allows for alternative justifications and alternative hierarchical edges. In particular, several outgoing justifications of a node n, which are not connected by hierarchical edges, are OR-alternatives. That is, to prove a node n, only the targets of one of these justifications have to be solved. Hence they represent alternative ways to tackle the same problem n. This describes the horizontal structure of a proof. Note further that we can share refinements: for instance, two abstract justifications may be refined by one and the same justification at lower levels.

Hierarchical edges are used to construct the vertical structure of a proof. This mechanism supports both recursive expansion and abstraction of proofs. For instance, in Fig. 2a, the edge from j_2 to j_1 indicates that j_2 refines j_1. The hierarchical edges distinguish between upper layer proof steps and their refinements at a more granular layer.



(a) PDS-node with all outgoing partially hierarchically ordered justifications, and j_1, j_4 in the set of alternatives. Justifications are depicted as boxes.



(b) PDS-node in the PDS-view obtained for the selected set of alternatives j_1, j_4.

Fig. 2: An example PDS and one of its PDS-views

A proof may be first conceived at a high level of abstraction and then *expanded* to a finer level of granularity. Vice versa, *abstraction* means the process of successively contracting fine-grained proof steps to more abstract proof steps. Furthermore, the PDS generally supports alternative and mutually incomparable refinements of one and the same upper layer proof step. This horizontal structuring mechanism—together with the possibility to represent OR-alternatives at the vertical level—provides very rich and powerful means to represent and maintain the proof attempts during the search for the final proof. In fact, such multidimensional proof attempts may easily become too complex for a human user, but since the user does not have to work simultaneously on different granularities of a proof, elaborate functionalities to access only selected parts of a PDS are helpful. They are required, for instance, for user-oriented presentation of a PDS, in which the user should be able to focus only on those parts of the PDS he is currently working with. At any time, the user can choose to see more details of some proof step or, on the contrary, he may want to see a coarse structure when he is lost in details and cannot see the wood for trees.

One such functionality is a *PDS-view* that extracts from a given PDS only a horizontal structure of the represented proof attempts, but with all its OR-alternatives. As an example consider the PDS fragments in Fig. 2.
The node n_1 in the fragment on the left has two alternative proof attempts with different granularities. The fragment on the right gives a PDS-view which results from selection of a certain granularity for each alternative proof attempt, respectively. The set of alternatives may be selected by the user to define the granularity on which he currently wants to inspect the proof. The resulting PDS-view is a slice plane through the hierarchical PDS and is—from a technical point of view—also a PDS, but without hierarchies, that is without hierarchical edges.

2.2 Searching for a Proof

In the following we shall look at our main mechanisms for actually finding a proof and we distinguish two basic modes, knowledge based, i.e. deliberative proof search and reactive proof search.

2.2.1 Knowledge-based Proof Search

ΩMEGA's main focus is on knowledge-based proof planning (Bundy, 1988, 1991; Melis and Siekmann, 1999a; Melis et al, 2007), where proofs are not conceived in terms of low-level calculus rules, but at a less detailed granularity, that is at a more abstract level, that highlights the main ideas and de-emphasises minor logical or mathematical manipulations of formulas. The motivation is to reduce the combinatorial explosion of the search space in classical automated theorem proving by providing means for a more global search control. Indeed, the search space in proof planning tends to be orders of magnitude smaller than at the level of calculus rules (Bundy, 2002; Melis and Siekmann, 1999b). Furthermore, a purely logical proof more often than not obscures its main mathematical ideas.

Knowledge-based proof planning is a paradigm in automated theorem proving, which swings the motivational pendulum back to the AI origins in that it employs and further develops many AI principles and techniques such as hierarchical planning, knowledge representation in frames and control rules, constraint solving, tactical theorem proving, and meta-level reasoning.

It differs from traditional search based techniques not least in its level of granularity: The proof of a theorem is planned at an abstract level where an *outline* of the proof is found first. This outline, that is, the abstract proof plan, can be recursively expanded to construct a proof within a logical calculus provided the expansion of the proof plan does not fail.

The building blocks of a proof plan are the plan operators, called *methods* which represent mathematical techniques familiar to a working mathematician. Another important feature is the separation of the knowledge of when to apply a certain technique from the technique itself, which is explicitly stored in *control rules*. Control rules cannot only reason about the current goals and assumptions, but also about the whole proof attempt so far.

Methods and control rules can employ external systems (for instance, a method may call one of the computer algebra systems) and make use of the knowledge in these systems. ΩMEGA's multi-strategy proof planner MULTI (Melis and Meier, 2000; Meier and Melis, 2005; Melis et al, 2007) searches for a plan using the acquired methods and strategies guided by the control knowledge in the control rules. In general, proof planning provides a natural basis for the integration of computational systems for both guiding the automated proof construction and performing proof steps.

Knowledge-based proof planning was successfully applied in many mathematical domains, including the domain of limit theorems (Melis and Siekmann, 1999a), which was proposed by Woody Bledsoe (Bledsoe, 1990) as a challenge to automated reasoning systems. The general-purpose planner makes use of the mathematical domain knowledge for ϵ-δ-proofs and of the guidance provided by declaratively represented control rules, which correspond to mathematical intuition about how to prove a theorem in a given situation. Knowledge based proof planning has also been applied to residue-classes problems (Meier et al, 2002b), and to plan "irrationality of $\sqrt[j]{l}$"-conjectures for arbitrary natural numbers j and l (Siekmann et al, 2003).

Methods, Control Rules, and Strategies

Methods were originally invented by Alan Bundy (Bundy, 1988) as tactics augmented with preconditions and effects, called *premises* and *conclusions*, respectively. A method represents a large inference of the conclusion from the premises based on the body of the tactic. The advantage of specifying the effects of a tactic are twofold: (i) the attached tactic need not be executed during the search, (ii) the specification of the tactic may contain additional constraints or control knowledge to restrict the search.

Knowledge-based proof planning expands on these ideas by focusing on encoding domain or problem-specific mathematical methods as proof-planning methods and additionally supports the explicit representation of control knowledge and strategic knowledge. For instance, consider the methods *hom1-1* in Fig. 3.

$$\frac{\boxed{P_1 : \forall Y.\ \Phi} \qquad \boxed{P_2 : \forall x.\ P(x) \Leftrightarrow \Psi} \qquad P_3 : P(C)}{\boxed{C : P(F(C))}}\ \textit{hom1-1}(\Phi_1, \Phi_2)$$

Appl. Cond.:	—
Completions:	$\langle \Phi_1 \leftarrow \text{termrploccs}(X, \Phi, C) \rangle$
	$\langle \Phi_2 \leftarrow \text{termrploccs}(X, \Phi, F(C)) \rangle$
Expansion:	$P_5 : \Phi_2$

Fig. 3: Method *hom1-1*

It encodes a planning operator that uses premises P_1, P_2, P_3 to prove the subgoal C. P_1, P_2, and C are annotated with control information stating in this case that they must be instantiated. Informally, the method describes the following proof situation: If f is a given function, p a defined predicate and the goal is to prove $p(f(c))$, then show $p(c)$ and use this to show $p(f(c))$. Note that P_5 is an open goal that does not occur in the specification and therefore does not directly enter the planning process. The later expansion process will insert p_5 as additional goal in the planning state and then call the planner to close it. To postpone the proof of a goal is an essential feature of methods and provides a means to structure the search space.

Control rules represent mathematical knowledge about how to proceed in the proof planning process. They can influence the planner's behaviour at choice points (such as deciding which goal to tackle next or which method to apply next) by preferring members of the corresponding list of alternatives (for instance, the list of possible goals or the list of possible methods). This way promising search paths are preferred and the search space can be pruned. An example of a control rule is shown in Fig. 4.

```
(control-rule prove-inequality
  (kind methods)
  (IF (and (goal-matches (REL A B))
      (in REL {<,>,<=,>=})))
  (THEN (prefer (TELLCS-B TELLCS-F, ASKCS-B, SIMPLIFY-B,
         SIMPLIFY-F, SOLVE*-B, COMPLEX-ESTIMATE-B,
         FACTORIALESTIMATE-B, SET-FOCUS-B))))
```

Fig. 4: Control Rule `prove-inequality`

Its IF-part checks whether the current goal is an inequality. If this is the case, it prefers the methods stated after the keyword prefer of the rule in the specified order. The general idea of this control rule is that a constraint should be simplified until it can be handled by the constraint solver, which collects constraints in goals and assumptions through the methods TELLCS-B and TELLCS-F.

Strategies encapsulate fixed sets of methods and control rules and, thus, tackle a problem by some mathematical standard that happens to be typical for this problem. The reasoning as to which strategy to employ for a given problem is an explicit choice point in MULTI. In particular the MULTI system can backtrack from a chosen strategy and commence search with different strategies. An example of a strategy is shown in Fig. 5.

```
(strategy SolveInequality
   (condition inequality-task)
   (algorithm PPlanner)
   (methods COMPLEXESTIMATE-B, TELLCS-B, TELLCS-F, SOLVE*-B ...)
   (crules prove-inequality, eager-instantiage ...)
   (termination no-inequalities)
   )
```

Fig. 5: Strategy SolveInequality

It is applicable for tasks whose formulas are inequalities or whose formulas can be reduced to inequalities. It comprises methods such as COMPLEXESTIMATE-B and TELLCS-B. It consists of control rules such as prove-inequality. The strategy terminates, if there are no further tasks containing inequalities.

Detailed discussions of ΩMEGA's method and control rule language can be found in (Meier, 2004; Meier et al, 2002a). A detailed introduction to proof planning with multiple strategies is given in (Melis and Meier, 2000; Meier and Melis, 2005) and more recently in (Melis et al, 2007).

2.2.2 Reactive Proof Search

The Ω-ANTS-system was originally developed to support interactive theorem proving (Benzmüller and Sorge, 1998). Later it was extended to a fully automated theorem prover (Benzmüller and Sorge, 2000; Sorge, 2001) and incorporated into the ΩMEGA-system. The basic idea is to encapsulate each inference into an agent, called an *inference ant*. All ants watch out for their applicability thus generating, in each proof situation, a ranked list of bids for their application. In this process, all inferences are uniformly viewed wrt. their arguments, that is, their premises, conclusions as well as other parameters. An inference is applicable if we have found a complete partial argument instantiation and the task of the Ω-ANTS system is to incrementally find complete partial argument instantiations. This starts with an empty partial argument instantiation and searches for instantiations for the missing arguments. This search is performed by separate, concurrent processes (software agents) which compute and report bids for possible instantiations. In other words, each inference ant is realised as a cooperating society of *argument ants*.

The Ω-ANTS architecture

The architecture consists of two layers. At the bottom layer, bids of possible instantiations of the arguments of individual inference are computed by argument ants. Each inference gets its own dedicated blackboard and one or several argument ants for each of its arguments. The role of each argument ant is to compute possible instantiations for its designated argument of the inference, and, if successful to record these as bids on the blackboard for this inference. The computations are carried out within the given proof context and by exploiting bids already present on the inference's blackboard, that is, argument instantiations computed by other argument ants working for the same rule. On the upper layer, the bids from the bottom layer that are applicable in the current proof state are accumulated and heuristically ranked by another process. The most promising bid on the upper layer is then applied to the central proof object and the data on the blackboards is cleared for the next round of computations.

Ω-ANTS employs resource-bounded reasoning to guide the search and provides facilities to define and modify the processes at runtime (Benzmüller and Sorge, 1999). This enables the controlled integration (for instance, by specifying time-outs) of full-fledged external reasoning systems such as automated theorem provers, computer algebra systems, or model generators into the architecture. The use of the external systems is modelled by inferences, usually one for each system. Their corresponding computations are encapsulated into one of the argument ants connected to this inference. For example, consider an inference encapsulating the application of an ATP:

$$\frac{\mathrm{p} : \Psi}{\mathrm{c} : \Phi}\ \mathrm{ATP}$$

$$\textbf{Appl. Cond.:}\ \Phi = \top$$

$$\textbf{Completions:}\ \langle \Psi \leftarrow \mathrm{ATP}(\Phi) \rangle$$

The inference contains an completion function that computes the value for its premise given a conclusion argument, that is, an open goal. This completion function is turned into an argument ant for the premise argument. Once an open goal is placed on the blackboard, this argument ant picks it up and applies the prover to it in a concurrent process. While the prover runs, other argument ants for other inferences may run in parallel and try to enable their application. Once the prover found a proof or a partial-proof, it is again written onto the blackboard and subsequently inserted into the proof object if the inference is applied. The semantics of the connections to external reasoners is currently hand-coded, but an ontology could be fruitfully employed like the one suggested in (Sutcliffe et al, 2004).

The advantage of this setup is that it enables proof construction by a collaborative effort of diverse reasoning systems. Moreover, the architecture provides a simple and general mechanism for integrating new reasoners into the system independently of other systems already present. Adding a new reasoning system to the architecture requires only to model it as an inference and to provide the argument ants and the inference ant for it. These ants then communicate with the blackboard by reading and writing subproblems to and from it, as well as writing proofs back to the blackboard in a standardised format.

Communication as a Bottleneck

A main disadvantage of our generic architecture is the communication overhead, since the results from the external systems have to be translated back and forth between their respective syntax and the language of the central proof object. Ω-ANTS has initially been rather inefficient: the case studies reported in (Benzmüller et al, 1999b) show that the larger part of the proof effort is sometimes spent on communication rather than on the actual proof search.

In order to overcome this problem, we devised a new method for the cooperation between two integrated systems via a single inference rule, first presented in (Benzmüller et al, 2005). This effectively cuts out the need to communicate via the central proof object.

However, direct bilateral integration of two reasoning system is difficult if both systems do not share representation formalisms that are sufficiently similar: implementing a dedicated inference for the combination of two particular reasoning systems is more cumbersome than simply integrating each system and its results into Ω-ANTS' central architecture. See (Benzmüller et al, 2007) for a detailed case study, which evaluates this approach for the cooperation between the higher-order resolution prover LEΩ (Benzmüller and Kohlhase, 1998) and a first-order theorem prover. The general idea is that LEΩ sends the subset of its clauses that do not contain any 'real' higher-order literals to a first-order theorem prover.

3 Knowledge as a Resource

There is a need to organise the different knowledge forms and determine which knowledge is available for which problem. Furthermore, there is a need to avoid redundant information for knowledge maintenance reasons. In the ΩMEGA system we use development graphs (Hutter, 2000; Autexier and Hutter, 2005) as a general mechanism to maintain inferences, strategies and control rules in the system (see Section 3.1). In Section 3.2 we describe how this knowledge can be formalised and included into the development graph. In Section 3.3 we present how inferences can be automatically synthesised from axioms and lemmas maintained in the development graph. Based on inferences, we describe how the knowledge for the planner can be automatically synthesised from inferences (Section 3.4) and Section 3.5 describes the mechanism to automatically generate a set of argument agents out of the inference descriptions.

3.1 Managing Mathematical Knowledge

The knowledge of the ΩMEGA system is organised in theories that are built up hierarchically by importing knowledge from lower theories via theory morphisms to upper layers. These theories and morphisms are organised respectively as nodes and edges of *development graphs* as implemented in the MAYA system (Autexier and Hutter, 2005) which is the central component of the ΩMEGA system that maintains information about the status of conjectures (unknown, proved, disproved, or in-progress) and controls which knowledge is available for which conjecture. MAYA supports the evolution of mathematical theories by a sophisticated mechanism for the *management of change*. Its main feature is to maintain the proofs already done when changing or augmenting the theories.

Each theory in the development graph contains standard information like the signature of its mathematical concepts and their formalisation with axioms, lemmas and theorems. The development graph stores other kinds of knowledge as well, such as specific inferences, strategies, control rules and information that links the symbols with their defining axioms as well as symbol orderings.

Each knowledge item is attached to a specific theory. To make it actually visible in all theories that are built on that theory, the development graph allows to specify how morphisms affect the knowledge item.

For each open lemma in a specific theory, the development-graph provides all knowledge items that can potentially be used for the lemma. It is up to the proof procedure to select the relevant parts and possibly transform them into a specific representation for the proof procedure.

3.2 Formalising Mathematical Knowledge

To accommodate a variety of input forms, the ΩMEGA system uses the OMDOC (Kohlhase, 2006) document format as a uniform interface for structured theories.

The OMDOC standard is an XML-language for **O**pen **M**athematical **Doc**uments, which includes structured theories modelled on development graphs and a proof representation formalism that is modelled on ΩMEGA's proof datastructure PDS (Autexier and Sacerdoti-Coen, 2006).

Structured theories are not the only knowledge forms we use: Inferences can be automatically synthesised from assertions (Section 3.3) and the required planner methods and agents can in turn be synthesised from inferences (Sections 3.4 and 3.5).

The development and encoding of proof methods by hand is a laborious and therefore we studied automatic learning techniques for this problem in collaboration with colleagues from the LEARNΩMATIC project (Jamnik et al, 2003).

When a number of proofs use a similar reasoning pattern, this pattern should be captured by a new method in proof planning and the LEARNΩMATIC system can now learn new methods automatically from a number of well chosen (positive) examples. Automated learning of proof methods is particularly interesting since theorems and their proofs exist typically in abundance,[3] while the extraction of methods from these examples is a major bottleneck of the proof planning methodology. The evaluation of the LEARNΩMATIC system showed that approach this leads to methods that make proofs shorter, reduces search and enables the roof planner to prove theorems that they could not before (Jamnik et al, 2003).

3.3 From Assertions to Inferences

How can we compute a set of inferences for arbitrary assertions? The intuitive idea is as follows; given the definition of $\subseteq$:

$$\forall U, V. U \subseteq V \Leftrightarrow (\forall x. x \in U \Rightarrow x \in V)$$

Reading the equivalence as two implications, this assertion results in the two inferences:

$$\frac{\begin{array}{c}[X \in U]\\ \vdots \\ p : X \in V\end{array}}{c : U \subseteq V}\,\textit{Def-}\subseteq \qquad\qquad \frac{p_1 : U \subseteq V \quad p_2 : X \in U}{c : X \in V}\,\textit{Def-}\subseteq$$

Appl. Cond.: X *new for* U *and* V **Appl. Cond.:** X *new for* U *and* V

where U, V, and X are meta-variables.

Another example is the definition of the limit of a function

$$\begin{array}{l}\forall f, a, l.\\ \quad \forall \epsilon. \epsilon > 0 \Rightarrow \exists \delta. \delta > 0 \Rightarrow \forall x. (0 <\mid x - a \mid \wedge \mid x - a \mid < \delta) \Rightarrow \mid f(x) - l \mid < \epsilon \\ \qquad \Rightarrow \lim_a f = l\end{array} \tag{3}$$

which can be turned into the inference

[3] For example, the testbed we developed for proof planning theorems about residue classes consists of more than 70000 theorems.

$$\frac{\begin{array}{c}[\epsilon > 0, D > 0, 0 < \mid x - A \mid, \mid x - A \mid < D]\\ \vdots \\ P : \mid F(x) - L \mid < \epsilon\end{array}}{C : \lim_{A} F = L} \tag{4}$$

Application Condition: $\mathsf{EV}(\epsilon, \{F, A, L\}) \wedge \mathsf{EV}(x, \{F, A, L, D\})$
Parameters: ϵ, x

where F, A, L, D are meta-variables, ϵ and x are parameters. $\mathsf{EV}(x, \{F, A, L, D\})$ is the application condition requiring that the parameter x should not occur in the instances of $\{F, A, L, D\}$ (i.e., x is an Eigenvariable wrt. the instances of F, A, L, and D).

The technique to obtain such inferences automatically from assertions (Autexier and Dietrich, 2006) follows the introduction and elimination rules of a natural deduction (ND) calculus (Gentzen, 1969). Given a formula, in a first phase the ND elimination rules are exhaustively applied to that formula collecting the *Eigenvariable* conditions as we go. This results in a set of inference descriptions with *Eigenvariable* conditions. In a second phase the premises of the inference descriptions are simplified by exhaustively applying ND introduction rules to the premises as well as to possible hypotheses obtained for the premises in that phase. The collected *Eigenvariable* conditions are of the form *"y new wrt. S"*, where y is the Eigenvariable and S is a list of constants and meta-variables in which y must not occur (including the symbols in the meta-variable substitutions). Checking these conditions by using the predicate $\mathsf{EV}(y, S)$ we compute inferences of the form

$$\frac{\begin{array}{ccc}[\mathcal{H}_1] & & [\mathcal{H}_n]\\ \vdots & & \vdots \\ P_1 & \ldots & P_n\end{array}}{C}\,\textbf{Parameters}\ (y_1, \ldots, y_n)$$

Application Condition: $\mathsf{EV}(y_1, S_1) \wedge \ldots \wedge \mathsf{EV}(y_m, S_m)$

for every assertion.

3.4 From Inferences to Planner Methods

Inferences are either operational representations of domain axioms, lemmas and theorems or user-defined, domain or problem specific mathematical methods. This may even include specialised computing and reasoning systems. Now, inferences can be applied in many ways (see Section 2.1.2), but not all of them contribute to the goal of the current proof plan. Rather, efficient (and controlled) search is only possible if we chose an appropriate subset of the many application directions. For example, suppose the current task is to show $A \subset B, x \in B \Rightarrow x \in A \vdash A = B$ and we are given the inference

$$\frac{P_1 : A \subset B \qquad P_2 : B \subset A}{C : A = B}$$

originating from the assertion $A = B \Leftrightarrow A \subset B \wedge B \subset A$. Suppose further that the proof plan requires to unfold the definitions in the goal first and then to use logical arguments to finish the proof. With respect to the first steps in the proof plan, only those application directions of this inference make sense, in which the conclusion C is instantiated, i.e. the following four partial argument instantiations: $\{P_1, P_2, C\}, \{P_1, C\}, \{P_2, C\}, \{C\}$. A convenient way to select the "right" subset is to specify implicitly the subset $\{ad \in AD | ad \models \Phi\}$ of the application directions AD by a property Φ, such that the application direction ad of interest satisfies it. In other words, Φ determines exactly those application directions which are compatible with the current meta level goal or the current mathematical technique.

In our example, we could specify the subset of interest by requiring that the partial argument instantiations are *backward*, where "backward" means that all conclusions of the inference are instantiated (here C). Another, equally appropriate way would be to characterise the subset of interest as those which reduce the target term with respect to a term ordering.

An inference augmented with the information about the application direction is called a *planning method*. However, given a set of planning methods there is no control information which ranks the inferences, i.e., which controls the choice in case several methods are applicable. This is done by the control rules (see Section 2.2.1) that are also maintained in the development graph and provided manually or are part of strategy descriptions.

3.5 From Inferences to Agents

Given an inference, we may wish to synthesise a society of agents as ants in Ω-ANTS in addition to proof planning methods. The problem is that such a society is "fragile" in the sense that removing one agent can result in a non-operational unit that cannot produce useful suggestions or any suggestions at all. Hence we must choose a sufficiently large set of agents such that for each agent there is another agent which produces partial argument instantiations required by the agent. But each agent allocates valuable resources (space and runtime). Thus creating all possible agents would deteriorate the system performance, as it would take too much time to compute any suggestion at all.

Our solution is to generate a so-called *agent creation graph*, whose nodes are equivalence classes on partial argument instantiations, and whose edges are all possible partial argument instantiation updates. A society of agents induces a subgraph of the agent creation graph by restricting the edges corresponding to the society of agents. Reachability of a node from the equivalence class of the empty partial argument instantiation in the induced subgraph means that partial argument instantiations for this equivalence class can be generated by the society of agents. The problem of generating an efficient society of agents such that all equivalence classes are reachable is then a *single source shortest path problem*, where we assign positive weights to the edges in the graph. This problem can then be solved by known algorithms (Dijkstra, 1959).

A comparison in (Dietrich, 2006) of some automatically generated units of argument agents with manually specified argument agents shows that they are almost identical. For details about the algorithm see (Dietrich, 2006; Autexier and Dietrich, 2006).

4 Specialised Computing and Reasoning Resources

Mathematical theorem proving requires a great variety of skills, hence it is desirable to have several systems with complementary capabilities, to orchestrate their use, and to integrate their results.

The situation is comparable to, say, a travel booking system that must combine different information sources, such as the search engines, price computation schemes, and the travel information in distributed (very) large databases, in order to answer a booking request. The information sources are distributed over the Internet and the access to such specialised travel information sources has to be planned, the results have to be combined and, finally there must be a consistency check of the time constraints.

In (Zimmer, 2008; Zimmer and Autexier, 2006) this methodology was transferred and applied to mathematical problem solving. The MATHSERV system plans the combination of several mathematical information sources (such as mathematical databases), computer algebra systems (CASs), and reasoning processes such as automated theorem provers (ATPs), constraint solvers (CSs) or model generation systems (MGs).

The MATHSERV system is based on the MATHWEB-SB network of mathematical services (Franke and Kohlhase, 1999; Zimmer and Kohlhase, 2002), which was the first approach for an open and modern software environment that enables modularisation, distribution and networking of mathematical services. This provided the infrastructure for building a society of software agents that render mathematical services by either encapsulating legacy deduction software or other functionalities. The software agents deliver their services via a common mathematical software bus in which a central broker agent provides routing and authentication information. Once the connection to a reasoning system has been established by the broker, the requesting client has to access the reasoning system directly via its API. The software bus and its associated reasoning systems were used not only within the field of automated theorem proving, but also for the semantic analysis of natural language (disambiguating syntactical constraints), verification tasks (proving a verification condition), and others, which resulted sometimes in several thousand theorems per day to be proven routinely for these external users.

The MATHSERV system extends the MATHWEB-SB's client-server architecture by semantic brokering of mathematical services and advanced problem solving capabilities to orchestrate the access to the reasoning systems. The key aspects of the MATHSERV framework are:

Problem-Oriented Interface: a more abstract communication level for MATHWEB-SB, such that general mathematical problem descriptions can be sent to MATHSERV which in turn returns a solution to that problem. Essentially, we moved from the *service* oriented interface of MATHWEB-SB to a *problem* oriented interface for MATHSERV.

Advanced Problem Solving Capabilities: Typically, a given problem cannot be solved by a single service but only by a combination of several services. In order to support the automatic selection and combination of existing services, the key idea is as follows: an ontology is used for the qualitative description of MATHWEB-SB services and *these descriptions are then used as AI planning operators*, in analogy to the proof planning approach. MATHSERV uses planning techniques (Carbonell et al, 1992; Erol et al, 1994) to automatically generate a plan that describes how existing services must be combined to solve a given mathematical problem.

We used external systems in the search for a proof in two ways within ΩMEGA: to provide a solution to a subproblem, or to give hints for the control of the search. In the first case, the call of a reasoning system is modelled as an inference rule and the output of the incorporated reasoning system is translated and inserted as a subproof into the PDS. This back-translation is necessary for interfacing systems that operate at different levels of granularity, and also for a human-oriented display and inspection of a partial proof. In the other case, where the external system is used to compute values that may be used to guide the search process, the system can be called by a completion function or from within control rules.

The following external systems were integrated and used in the ΩMEGA system over the years:

Computer Algebra Systems (CAS) provide symbolic computation, which can be used to compute hints to guide the proof search (such as witnesses for existential variables), or, second, to perform some complex algebraic computation such as to normalise or simplify terms. In the latter case the symbolic computation is directly translated into proof steps in ΩMEGA. CASs are integrated via the transformation and translation module SAPPER (Sorge, 2000). Currently, ΩMEGA uses the systems MAPLE (Char et al, 1992) and GAP (Schönert et al, 1995).

Automated Deduction Systems (ATP) are used to solve subgoals, currently the first-order provers BLIKSEM (de Nivelle, 1999), EQP (McCune, 1997), OTTER (McCune, 1994), PROTEIN (Baumgartner and Furbach, 1994), SPASS (Weidenbach et al, 1999), WALDMEISTER (Hillenbrand et al, 1999), the higher-order systems TPS (Andrews et al, 1996), and LEΩ (Benzmüller and Kohlhase, 1998; Benzmüller, 1999), and VAMPIRE (Riazanov and Voronkov, 2001). The first-order ATPs were connected via TRAMP (Meier, 2000), which is a proof transformation system that transforms resolution-style proofs into assertion-level ND-proofs which were then integrated into ΩMEGA's PDS. The TPS system generates ND-proofs directly, which could then be further processed and checked with little transformational effort (Benzmüller et al, 1999a).

Model Generators (MG) provide either witnesses for free (existential) variables, or counter-models, which show that some subgoal is not a theorem. ΩMEGA used the model generators SATCHMO (Manthey and Bry, 1988) and SEM (Zhang and Zhang, 1995).

Constraint Solver (CS) construct mathematical objects with theory-specific properties as witnesses for free (existential) variables. Moreover, a constraint solver can reduce the proof search by checking for inconsistencies of constraints. ΩMEGA employed $\mathcal{C}o\mathcal{SIE}$ (Melis et al, 2000; Zimmer and Melis, 2004), a constraint solver for inequalities and equations over the field of real numbers.

Automated Theory Formation systems (ATF) explore mathematical theories and search for new properties. The HR system is an ATF system in the spirit of Doug Lenat's AM, which conjectures mathematical theories given empirical data (Colton, 2002). ΩMEGA used the HR system to provide instances for meta-variables that satisfy some required properties. The MATHSAID system proves and identifies theorems (lemmas, corollaries, etc.) from a given set of axioms and definitions (McCasland et al, 2006). MATHSAID was used by ΩMEGA to derive interesting lemmas for given mathematical theories which would enable the ATPs to prove theorems they could not prove without these lemmas.

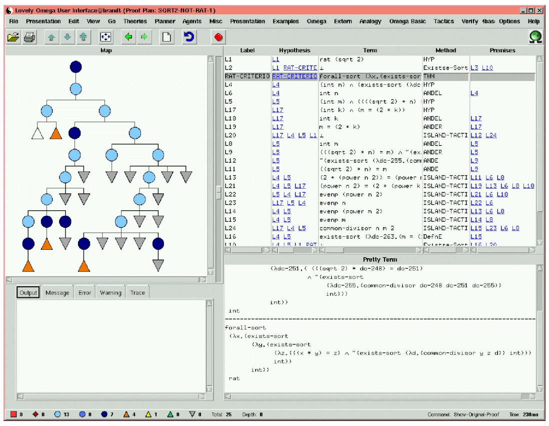
5 ΩMEGA as an adaptive Resource

If a mathematical assistance system is to be used as a resource by other systems as well as by users with different skills and backgrounds we have to redesign the architecture of the system to make it adaptive. We present the research and development for the interaction with a human-users in Section 5.1 and discuss the interaction with other software systems in Section 5.2.

5.1 Adaptation to Users with different Skills

The OMEGA research group early addressed the interaction between proof assistant system and human user. In a first stage we developed an elaborate graphical user interface LΩUI (Siekmann et al, 1999) (see Fig. 6a).

The three inter-connected windows present the shape of the central proof tree (left), information about the nodes in the tree (upper right), and the pretty printing of the complete formula of a selected node (lower right). There is also support to switch between different levels of granularity at which a proof can be presented and it is also possible to browse through mathematical theories in an HTML-like viewer. These functionalities were targeted towards a user, who has no knowledge about the

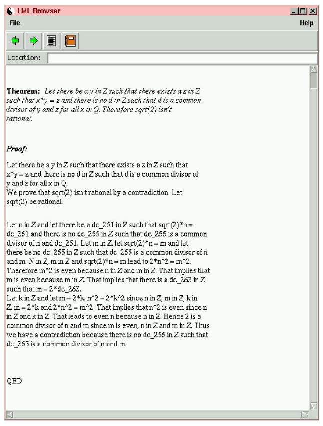

Lovely Omega User Interface@brandt (Proof Plan: SQRT2-NOT-RAT-1)
File Presentation Edit View Go Theories Planner Agents Misc Presentation Examples Omega Extern Analogy Omega Basic Tactics Verify Ωbase Options Help
Map
Output Message Error Warning Trace
Total: 25 Depth: 0 Command: Show-Original-Proof Time: 230ms

Label	Hypothesis	Term	Method	Premises
L1	L1	rat (sqrt 2)	HYP	
L2	L1 RAT-CRITE	⊥	Existse-Sort	L3 L10
RAT-CRITERIO	RAT-CRITERIO	forall-sort (λx.(exists-sor	THM	
L4	L4	(int n) ∧ (exists-sort (λdc	HYP	
L6	L4	int n	ANDEL	L4
L5	L5	(int m) ∧ ((((sqrt 2) * n)	HYP	
L17	L17	(int k) ∧ (m = (2 * k))	HYP	
L18	L17	int k	ANDEL	L17
L19	L17	m = (2 * k)	ANDER	L17
L20	L17 L4 L5 L1	⊥	ISLAND-TACTI	L12 L24
L8	L5	int m	ANDEL	L5
L9	L5	(((sqrt 2) * n) = m) ∧ ~(ex	ANDER	L5
L12	L5	~(exists-sort (λdc-255.(com	ANDE	L9
L11	L5	((sqrt 2) * n) = m	ANDE	L9
L13	L4 L5	(2 * (power n 2)) = (power	ISLAND-TACTI	L11 L6 L8
L21	L4 L5 L17	(power n 2) = (2 * (power k	ISLAND-TACTI	L19 L13 L6 L8 L18
L22	L5 L4 L17	evenp (power n 2)	ISLAND-TACTI	L21 L6 L18
L23	L17 L5 L4	evenp n	ISLAND-TACTI	L22 L6
L14	L4 L5	evenp (power m 2)	ISLAND-TACTI	L13 L6 L8
L15	L4 L5	evenp m	ISLAND-TACTI	L14 L8
L24	L17 L4 L5	common-divisor n m 2	ISLAND-TACTI	L15 L23 L6 L8
L16	L4 L5	exists-sort (λdc-263.(m = (	DefnE	L15
L10	L4 L5 L1 RAT	⊥	Existse-Sort	L16 L20

Pretty Term

```
          (λdc-251.( (((sqrt 2) * dc-248) = dc-251)
                   ∧ ~(exists-sort
                       (λdc-255.(common-divisor dc-248 dc-251 dc-255))
                       int)))
          int))
 int
--------------------------------------------------------------------------------
forall-sort
 (λx.(exists-sort
      (λy.(exists-sort
           (λz.(((x * y) = z) ∧ ~(exists-sort (λd.(common-divisor y z d)) int)))
           int))
      int))
 rat
```

(a) The linked proof window views



Theorem: *Let there be a y in Z such that there exists a z in Z such that x*y = z and there is no d in Z such that d is a common divisor of y and z for all x in Q. Therefore sqrt(2) isn't rational.*

Proof:

Let there be a y in Z such that there exists a z in Z such that x*y = z and there is no d in Z such that d is a common divisor of y and z for all x in Q.
We prove that sqrt(2) isn't rational by a contradiction. Let sqrt(2) be rational.

Let n in Z and let there be a dc_251 in Z such that sqrt(2)*n = dc_251 and there is no dc_255 in Z such that dc_255 is a common divisor of n and dc_251. Let m in Z, let sqrt(2)*n = m and let there be no dc_255 in Z such that dc_255 is a common divisor of n and m. N in Z, m in Z and sqrt(2)*n = m lead to 2*n^2 = m^2. Therefore m^2 is even because n in Z and m in Z. That implies that m is even because m in Z. That implies that there is a dc_263 in Z such that m = 2*dc_263.
Let k in Z and let m = 2*k. n^2 = 2*k^2 since n in Z, m in Z, k in Z, m = 2*k and 2*n^2 = m^2. That implies that n^2 is even since n in Z and k in Z. That leads to even n because n in Z. Hence 2 is a common divisor of n and m since m is even, n in Z and m in Z. Thus we have a contradiction because there is no dc_255 in Z such that dc_255 is a common divisor of n and m.

QED

(b) The natural language presentation of proofs with *P.rex*

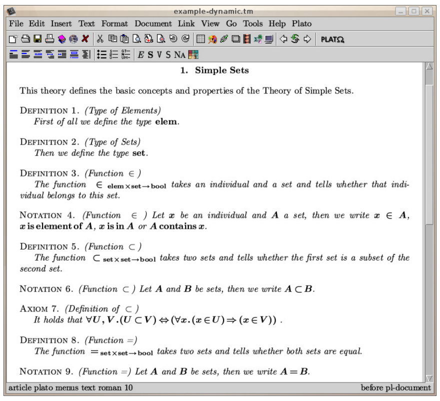
Fig. 6: LΩUI: The first Graphical User Interface of the mathematical assistant ΩMEGA

actual implementation and the programming language, but who is familiar with the main concepts of formal logic, natural deduction proofs, proof planning methods and tactics.

Also in the early 1980s, members of the group began to research the presentation of proofs in a text-book style form (see Fig. 6b), which does not pre-suppose skills in formal logic from the user. The translation of resolution proofs into natural deduction and the subsequent restructuring techniques for an improved presentation were early results (Lingenfelder, 1989, 1990). Based on these developments Xiarong Huang developed the PROVERB system (Huang, 1996), a landmark at its time, which translated these ND-proofs into well-structured natural language texts. Today we use the *P.rex*-system (Fiedler, 1999, 2001a,b), which is based on PROVERB, but presents the proof in a user-adaptive style, i.e. the mode of presentation and abstraction is relative to the skills of the user. The quality of the proof presentation generated by the *P.rex*-system is still a corner-stone in the area of proof presentation and the overall development in this field within the last three decades is the subject of the forthcoming textbook (Siekmann et al, to appear).

More recent work on human interaction with the system followed a different approach to make it more acceptable to the mathematical community. The mathematical assistance system must be integrated with the software that the users already employ, like standard text processing systems (such as LaTeX) for the preparation of documents. TeXMACS (van der Hoeven, 2001) is a scientific text-editor that provides professional type-setting and supports authoring with powerful macro definition facilities like those in LaTeX, but the user works on the final document ("What you see is what you get", WYSIWYG). As a first step we integrated the ΩMEGA system into TeXMACS using the generic mediator PLATΩ (Wagner et al, 2006). In this setting the formal content of a document is amenable to machine processing, without imposing any restrictions on how the document is structured and which language is used in the document. The PLATΩ system (Wagner, 2006) developed in the DFG-project VERIMATHDOC transforms the representation of the formal content of a document into the representation used in a proof assistance system and maintains the consistency between the two representations throughout potential changes.

In turn, the ΩMEGA system provides its support now transparently within the text-editor $\text{T}_\text{E}\text{X}_{\text{MACS}}$. Fig. 7 shows typical example documents on the screen:


example-dynamic.tm
File Edit Insert Text Format Document Link View Go Tools Help Plato
PLATΩ
E S V S NA

1. Simple Sets

This theory defines the basic concepts and properties of the Theory of Simple Sets.

DEFINITION 1. *(Type of Elements)*
First of all we define the type **elem**.

DEFINITION 2. *(Type of Sets)*
Then we define the type **set**.

DEFINITION 3. *(Function* $\in$ *)*
The function $\in_{\mathbf{elem}\times\mathbf{set}\to\mathbf{bool}}$ *takes an individual and a set and tells whether that individual belongs to this set.*

NOTATION 4. *(Function* $\in$ *) Let* x *be an individual and* A *a set, then we write* $x \in A$, x **is element of** A, x **is in** A *or* A **contains** x.

DEFINITION 5. *(Function* $\subset$ *)*
The function $\subset_{\mathbf{set}\times\mathbf{set}\to\mathbf{bool}}$ *takes two sets and tells whether the first set is a subset of the second set.*

NOTATION 6. *(Function* $\subset$ *) Let* A *and* B *be sets, then we write* $A \subset B$.

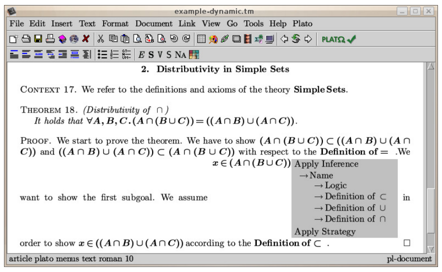
AXIOM 7. *(Definition of* $\subset$ *)*
It holds that $\forall U, V.(U \subset V) \Leftrightarrow (\forall x.(x \in U) \Rightarrow (x \in V))$.

DEFINITION 8. *(Function* $=$*)*
The function $=_{\mathbf{set}\times\mathbf{set}\to\mathbf{bool}}$ *takes two sets and tells whether both sets are equal.*

NOTATION 9. *(Function* $=$*) Let* A *and* B *be sets, then we write* $A = B$.

article plato menus text roman 10 before pl-document

(a)



2. Distributivity in Simple Sets

CONTEXT 17. We refer to the definitions and axioms of the theory **Simple Sets**.

THEOREM 18. *(Distributivity of* $\cap$ *)*
It holds that $\forall A, B, C.(A \cap (B \cup C)) = ((A \cap B) \cup (A \cap C))$.

PROOF. We start to prove the theorem. We have to show $(A \cap (B \cup C)) \subset ((A \cap B) \cup (A \cap C))$ and $((A \cap B) \cup (A \cap C)) \subset (A \cap (B \cup C))$ with respect to the **Definition of** $=$ .We want to show the first subgoal. We assume $x \in (A \cap (B \cup C))$ in order to show $x \in ((A \cap B) \cup (A \cap C))$ according to the **Definition of** $\subset$. □

article plato menus text roman 10 pl-document

(b)

Fig. 7: The user perspective of the support offered by ΩMEGA inside the text-editor $\text{T}_\text{E}\text{X}_{\text{MACS}}$ via PLATΩ.

Fig. 7a shows how the author can formalise mathematics: Based on the formal representation obtained by PLATΩ, the ΩMEGA system provides its support context-sensitively as a menu inside the text-editor. Fig. 7b shows such a menu generated by the system, that displays the different assertions which can be applied in the actual proof situation. The proof parts generated by ΩMEGA are patched into the document using natural language patterns. Current work is concerned with adapting the proof presentation techniques as used in *P.rex* to this setting. More details about that integration and the PLATΩ system can be found in (Wagner, 2006; Wagner et al, 2006).

All of this required the following changes in the architecture of the system: First, we need a clean interface with the text-editor $\text{T}_\text{E}\text{X}_{\text{MACS}}$. The role of this interface is to establish and maintain the correspondence of the objects in the document and their counter-parts within ΩMEGA. Based on the development graph, the definitions, axioms, and theorems in a $\text{T}_\text{E}\text{X}_{\text{MACS}}$ document are grouped into theories and they have a one-to-one correspondence to the development graph structure. The notion of a "PDS view" (Section 2.1.3) is the key prerequisite to consistently link the proof in the document with the respective part of the much more elaborate proof representation PDS in ΩMEGA: Each manually written proof step in the document is modelled as a proof sketch in the PDS, which has to be verified later-on.

Furthermore, the author of a document usually writes many different proofs for different theorems in different theories before the document is finished. This corresponds to multiple, parallel proof attempts in ΩMEGA: the development graph maintains the multiple ongoing proof attempts and determines which assertions in the document are visible for which proof attempt.

Having that infrastructure in place was the key to turn the ΩMEGA system into a server, that can provide mathematical assistance services for multiple documents in parallel sessions.

A second issue is that the author changes his document usually many times. Hence the proof assistance system has to be able to deal efficiently with non-monotonic changes not only inside a theory but also within the proofs. For instance, deleting an axiom from a theory should result in pruning or at least invalidating proof steps in all proofs that relied on that axiom. Furthermore, if by such an action a proof of some theorem is invalidated, then all other proof steps that used this theorem must be flagged and invalidated in turn. The immediate "solution", i.e. to automatically re-execute all proof procedures, is not an option in this setting: it would be too slow and short response times are an issue when the author of the document has to wait when "simply" deleting an axiom. Furthermore, re-executing the proof procedures may generate different proofs: since the proofs within ΩMEGA have to be synchronised with the proofs in the text-editor, this may result in drastic invasive (fully automatic) rearrangements of the document. Such a system behaviour would most certainly jeopardise the acceptance of the system.

For these reasons we integrated a sophisticated and fine granular truth-maintenance system, which tracks all dependencies between elements of a theory and their use in other theories and proofs (see (Autexier et al, 2008) for details).

5.2 Adaptation to different Software Systems

Just as ΩMEGA is used now as a subsystem within $\TeX_{\text{MACS}}$ it could be used by other software systems such as a program verification tool or a software development platform. Yet another application area we are currently working on is the integration of ΩMEGA into ACTIVEMATH, an e-learning system for mathematics (E. Melis, 2004).

More specifically, the DIALOG project (Benzmüller et al, 2003) studies natural language-based tutorial dialogs when teaching how to prove a theorem. Within a tutorial dialogue, the student has to prove a theorem interactively with the system. The system provides feedback to the student's input, corrects faulty steps and aids the student in finding a solution, with the overall goal to convey specific concepts and techniques of a given mathematical domain.

Due to the flexible and unpredictable nature of a tutorial dialogue it is necessary to dynamically process and analyse the informal input to the system, including linguistic analysis of the informal input to the system, evaluation of utterances in terms of *soundness*, *granularity* and *relevance*, and *ambiguity resolution* at all levels of processing. ΩMEGA is used to (i) represent the mathematical theory within which the proof exercise is carried out, i.e. the definitions, axioms, and theorems of the mathematical domain, (ii) to represent the ongoing proof attempts of the student, in particular the management of ambiguous proof states resulting from underspecified or ambiguous proof attempts, (iii) to maintain the mathematical knowledge the student is allowed to use and to react to changes of this knowledge, and (iv) to reconstruct intermediate steps necessary to verify a step entered by the student, thereby resolving ambiguity and underspecification.

The main problem in such a setting is the high-level and informal nature of human proofs: a classical automated deduction system is of little help here and these developments are only possible now, because of the high-level proof representation and proof planning techniques.

5.2.1 Checking the Correctness

The proof steps entered by the student are statements the system has to analyse with respect to its correctness. A proof step ideally introduces new hypotheses and/or new subgoals along with some justification how they have been obtained. If this information is complete and correct, the verification amounts to a simple check. However, in a tutorial setting, this is not the typical situation. The more likely and, from our point of view, more interesting case is that the statement is incomplete or faulty. Note that an incomplete proof step is not necessarily faulty: when writing proofs, humans typically omit information considered unimportant or trivial. Simply noting that a proof step is false or just incomplete is not a useful hint for the student, so we need a more detailed analysis. If no justification is given, but the hypotheses and subgoals can be correctly derived, the missing justification has to be computed. Conversely, if there is a justification, but the hypothesis or subgoal are missing, the missing parts should be returned by the system. If one of them is false, the system should return a corrected one.

In the sequel we show some typical phenomena extracted from a corpus of tutorial dialogues collected in the Wizard-of-Oz experiments between students and experienced math teachers (Benzmüller et al, 2006). Fig. 8 shows excerpts from collected dialogues, where the tutor's statements are marked with a capital **T** and the student's utterances with a capital **S**.

Underspecification: The proof step entered by the student is often not fully specified and information may be missing. Utterance **S1** in Fig. 8 is an example of this underspecification which appear throughout the corpus. The proof step in **S1** includes the application of set extensionality, but the rule is not stated explicitly. Also the student does not say which of the two subgoals introduced by set extensionality he is now proving, nor does he specify that there is a second subgoal. Further, the number of steps needed to reach this proof state is not given. Part of

T: Please prove $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$
S1: let $(x, y) \in (R \circ S)^{-1}$
T2: correct
S3: hence $(y, x) \in (S \circ R)$
T4: incorrect

S1a: we consider the subgoals $(R \circ S)^{-1} \subset S^{-1} \circ R^{-1}$ and $(R \circ S)^{-1} \supset S^{-1} \circ R^{-1}$
S1b: first, we consider the subgoal $(R \circ S)^{-1} \subset S^{-1} \circ R^{-1}$

Fig. 8: **left:** Example dialog between a tutor (**T**) and a student (**S**). **right:** Two alternative ways of how the student started to solve the exercise

the task of analysing such steps is to instantiate the missing information so that the formal proof object is complete.

Incomplete Information: In addition to issues of underspecification, there may be crucial information missing for the formal correctness analysis. For instance the utterance **S1** is clearly a contribution to the proof, but since the step only introduces a new variable binding, there is no assertion whose truth value can be checked. However, anticipating that the student wants to use the subset definition $A \subset B \Leftrightarrow x \in A \Rightarrow x \in B$ allows us to determine that the new variable binding is useful. Utterance **S1b** is also a correct contribution, but the second subgoal is not stated. This is however necessary in order to verify that the subset relation is part of the equality of sets.

These examples show that the verification in this scenario is not simply a matter of checking logical correctness, but we must take the proof context into account.

Ambiguity: Ambiguity pervades all levels of the analysis of natural language and mathematical expressions. Even in fully specified proof steps an element of ambiguity may remain. For example in any proof step which follows **S1a**, we don't know which subgoal the student has decided to work on. Also, when students state formulas without natural language expressions, such as "hence" or "conjecture", it is not clear whether the formula is a newly derived fact or a newly introduced conjecture. Again, this type of ambiguity can only be resolved in the context of the current proof. When no resolution is possible, the ambiguity must be propagated and this must be done by maintaining multiple parallel interpretations, which are retained until enough information is available later on in the proof attempt.

5.2.2 Cognitive Proof States

A well-known phenomenon with underspecified or faulty proof steps, is that there is in general more than one reasonable reconstruction. Each reconstruction directly influences the analysis of the subsequent proof step, that is, a subsequent step can be classified to be correct with respect to one reconstruction, but not with respect to another. Hence it is necessary to determine and maintain all possible reconstructions, which we call *cognitive proof states*. Ambiguities which can not immediately be resolved are propagated as parallel cognitive proof states until enough information is available for their resolution.

Technically, the PDS is used to simultaneously represent all of the possible cognitive proof states of the student, each represented by an agenda. Initially, there is only one cognitive proof state, containing the initial task $\mathsf{T} = \Gamma \vdash \Delta$ where Γ denotes the assumptions and Δ is the subgoal to be shown.

Updating the Cognitive Proof States

Given a set of possible cognitive proof states and a preprocessed utterance s, all possible successor states have to be determined, which are consistent with the utterance s. Each utterance is possibly—but not necessarily—annotated with information about whether the step represents a new lemma or whether it is supposed to contribute to the overall proof.

For each given cognitive proof state, we determine the successor states that are consistent with the utterance s. If no such successor state can be found this cognitive state is deleted. If several, alternative successor states can be found, i.e. the utterance s is ambiguous, they replace the given cognitive state.

That procedure also resolves ambiguities introduced in previous proof steps by deleting all cognitive states that are no longer consistent with the current utterance s. If all cognitive proof states are deleted, i.e. no successor state is found for any of the given cognitive states, the step is classified as *incorrect*.

The overall result is a confirmation of whether the step could be verified, along with the side-effect that the PDS has been updated to contain exactly the possible cognitive proof states resulting from the performance of the step. More details about the update process are given in (Dietrich and Buckley, 2007; Buckley and Dietrich, 2007).

5.2.3 Evaluation

We tested our proof-step checking component on a little less than 20 tutorial dialogs from a Wizard-of-Oz experiment, where students had to solve exercises on binary relations. Of 116 correct steps, 113 (97.4%) were correctly verified and we correctly classify 141 out of 144 steps (97.9%) as correct or wrong. The verification failed for the remaining three steps. The average checking time was around 4 seconds on a standard PC, the longest checking time has been 30 seconds. The main reason for the efficiency lies in the fact that we directly search at the assertion level, which makes a small search depth sufficient to verify correct proof steps.

6 Future Research

We now want to improve the system quality of ΩMEGA, such that we can train users to author documents with formal logical content. The ΩMEGA system now provides an adequate environment for this endeavour, because its high-level proof representation and proof planning techniques presuppose little knowledge—the main hurdle which typically hampers the use of such systems. Furthermore, ΩMEGA's capabilites to adapt to different users and usages provides a basis for the integration into standard text preparation systems and e-learning environments.

First we want to support authoring and maintenance of documents with formal logical content such that the author can formulate new concepts, conjectures and proofs in a document. Furthermore, we want to integrate other modalities like diagrams both to describe mathematical content and use it within mathematical proof.

Secondly, we want to further increase usability of formal reasoning tools by further developing the logical foundations of assertion-level proofs and automate proof search either by proof planning directly on the assertion-level or by transforming proofs obtained from classical automated deduction systems; a key question here is how to characterise and search for "good" proofs. Furthermore, we plan to automate proof search in large, structured theories, where, to date, human guidance of the proof procedures is indispensable, even for theorems that are simple by human standards. We will research how to exploit the structures in large theories not only to search for proofs but also to synthesise new interesting knowledge using automated theory formation (McCasland and Bundy, 2006).

Finally, we want to support the training of students in using formal reasoning tools. Rather than teaching students mathematical proof by forcing them to do a proof in a typical formal calculus, we want to allow the student to freely build any valid proof of the theorem at hand. On the tutoring side, this gives the freedom to adapt the tutoring to the student's skills: less experienced students will be taught more rigid proof styles that come close to the proof style enforced by classical formal calculi, while this is not imposed for more experienced students. In this context, we will further develop domain-independent criteria to dynamically evaluate the correctness, granularity and relevance of user uttered proof steps, provide domain-independent and domain-specific didactic strategies exploiting the dynamic proof step analysis capabilities of the ΩMEGA system, and exploit them to generate useful hints for the student.

References

Andrews P, Bishop M, Issar S, Nesmith D, Pfenning F, Xi H (1996) TPS: A theorem proving system for classical type theory. Journal of Automated Reasoning 16(3):321–353

Autexier S (2005) The CORE calculus. In: Nieuwenhuis R (ed) Proceedings of the 20th International Conference on Automated Deduction (CADE-20), Springer, Tallinn, Estonia, LNAI, vol 3632

Autexier S, Dietrich D (2006) Synthesizing proof planning methods and oants agents from mathematical knowledge. In: Borwein J, Farmer B (eds) Proceedings of MKM'06, Springer, LNAI, vol 4108, pp 94–109

Autexier S, Hutter D (2005) Formal software development in MAYA. In: Hutter D, Stephan W (eds) Festschrift in Honor of J. Siekmann, LNAI, vol 2605, Springer

Autexier S, Sacerdoti-Coen C (2006) A formal correspondence between omdoc with alternative proofs and the lambdabarmumutilde-calculus. In: Borwein J, Farmer B (eds) Proceedings of MKM'06, Springer, LNAI, vol 4108, pp 67–81

Autexier S, Benzmüller C, Dietrich D, Meier A, Wirth CP (2006) A generic modular data structure for proof attempts alternating on ideas and granularity. In: Kohlhase M (ed) Proceedings of the 5th International Conference on Mathematical Knowledge Management (MKM'05), Springer, LNAI, vol 3863, pp 126–142

Autexier S, Benzmüller C, Dietrich D, Wagner M (2008) Organisation, transformation, and propagation of mathematical knowledge in omega. submitted

Avenhaus J, Kühler U, Schmidt-Samoa T, Wirth CP (2003) How to prove inductive theorems? QUODLIBET! In: Proc. of the 19th Int. Conf. on Automated Deduction (CADE-19), Springer, no. 2741 in LNAI, pp 328–333

Baumgartner P, Furbach U (1994) PROTEIN, a PROver with a Theory INterface. In: Bundy A (ed) Proceedings of the 12th Conference on Automated Deduction, Springer, no. 814 in LNAI, pp 769–773

Benzmüller C (1999) Equality and extensionality in higher-order theorem proving. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany

Benzmüller C, Kohlhase M (1998) LEO – a higher-order theorem prover. In: Kirchner C, Kirchner H (eds) Proceedings of the 15th International Conference on Automated Deduction (CADE-15), Springer, Lindau, Germany, no. 1421 in LNAI, pp 139–143

Benzmüller C, Sorge V (1998) A blackboard architecture for guiding interactive proofs. In: Giunchiglia F (ed) Proceedings of 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98), Springer, no. 1480 in LNAI

Benzmüller C, Sorge V (1999) Critical agents supporting interactive theorem proving. In: Borahona P, Alferes JJ (eds) Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA'99), Springer, Evora, Portugal, no. 1695 in LNAI, pp 208–221

Benzmüller C, Sorge V (2000) Ωants – An open approach at combining Interactive and Automated Theorem Proving. In: Kerber M, Kohlhase M (eds) 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000), AK Peters

Benzmüller C, Bishop M, Sorge V (1999a) Integrating TPS and ΩMEGA. Journal of Universal Computer Science 5:188–207

Benzmüller C, Jamnik M, Kerber M, Sorge V (1999b) Agent based mathematical reasoning. Electronic Notes in Theoretical Computer Science, Elsevier 23(3):21–33

Benzmüller C, Fiedler A, Gabsdil M, Horacek H, Kruijff-Korbayova I, Pinkal M, Siekmann J, Tsovaltzi D, Vo BQ, Wolska M (2003) Tutorial dialogs on mathematical proofs. In: Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems, Acapulco, Mexico, pp 12–22

Benzmüller C, Sorge V, Jamnik M, Kerber M (2005) Can a higher-order and a first-order theorem prover cooperate? In: Baader F, Voronkov A (eds) Proceedings of the 11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR), Springer, no. 3452 in LNAI, pp 415–431

Benzmüller C, Horacek H, Lesourd H, Kruijff-Korbayova I, Schiller M, Wolska M (2006) A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material. In: Proceedings of International Conference on Language Resources and Evaluation (LREC 2006), ELDA, Genova, Italy

Benzmüller C, Sorge V, Jamnik M, Kerber M (2007) Combined reasoning by automated cooperation. Journal of Applied Logic In Print

Bledsoe W (1990) Challenge problems in elementary calculus. Journal of Automated Reasoning 6:341–359

Buckley M, Dietrich D (2007) Integrating Task Information into the Dialogue Context for Natural Language Mathematics Tutoring. In: Medlock B, Ó Séaghdha D (eds) Proceedings of the 10th Annual CLUK Research Colloquium, Cambridge, UK

Bundy A (1988) The use of explicit plans to guide inductive proofs. In: (Lusk and Overbeek, 1988), pp 111–120

Bundy A (1991) A science of reasoning. In: J-L Lasser GP (ed) Computational Logic: Essays in Honor of Alan Robinson, MIT Press, pp 178–199

Bundy A (2002) A critique of proof planning. In: Computational Logic: Logic Programming and Beyond (Kowalski Festschrift), LNAI, vol 2408, Springer, pp 160–177

Carbonell J, Blythe J, Etzioni O, Gil Y, Joseph R, Kahn D, Knoblock C, Minton S, Pérez MA, Reilly S, Veloso M, Wang X (1992) PRODIGY 4.0: The Manual and Tutorial. CMU Technical Report CMU-CS-92-150, Carnegie Mellon University

Char B, Geddes K, Gonnet G, Leong B, Monagan M, Watt S (1992) First leaves: a tutorial introduction to Maple V. Springer

Cheikhrouhou L, Sorge V (2000) PDS – a three-dimensional data structure for proof plans. In: Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)

Colton S (2002) Automated Theory Formation in Pure Mathematics. Distinguished Dissertations, Springer

Dietrich D (2006) The task-layer of the ΩMEGA system. Diploma thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany

Dietrich D, Buckley M (2007) Verification of Proof Steps for Tutoring Mathematical Proofs. In: Luckin R, Koedinger KR, Greer J (eds) Proceedings of the 13th International Conference on Artificial Intelligence in Education, IOS Press, Los Angeles, USA, vol 158, pp 560–562

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numerische Mathematik 1:269–271

E Melis JS (2004) Activemath: An intelligent tutoring system for mathematics. In: Rutkowski L, Siekmann J, Tadeusiewicz R, Zadeh L (eds) Seventh International Conference 'Artificial Intelligence and Soft Computing' (ICAISC), Springer-Verlag, LNAI, vol 3070, pp 91–101

Erol K, Hendler J, Nau D (1994) Semantics for hierarchical task network planning. Tech. Rep. CS-TR-3239, UMIACS-TR-94-31, Computer Science Department, University of Maryland

Fiedler A (1999) Using a cognitive architecture to plan dialogs for the adaptive explanation of proofs. In: Dean T (ed) Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann, Stockholm, Sweden, pp 358–363

Fiedler A (2001a) Dialog-driven adaptation of explanations of proofs. In: Nebel B (ed) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann, Seattle, WA, pp 1295–1300

Fiedler A (2001b) User-adaptive proof explanation. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany

Franke A, Kohlhase M (1999) System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In: (Ganzinger, 1999)

Ganzinger H (ed) (1999) Proceedings of the 16th Conference on Automated Deduction, no. 1632 in LNAI, Springer

Gentzen G (1969) The Collected Papers of Gerhard Gentzen (1934-1938). Edited by Szabo, M. E., North Holland, Amsterdam

Hillenbrand T, Jaeger A, Löchner B (1999) System description: Waldmeister — improvements in performance and ease of use. In: (Ganzinger, 1999), pp 232–236

van der Hoeven J (2001) GNU TeXmacs: A free, structured, wysiwyg and technical text editor. In: Actes du congrès Gutenberg, Metz, no. 39-40 in Actes du congrès Gutenberg, pp 39–50

Huang X (1996) Human Oriented Proof Presentation: A Reconstructive Approach. No. 112 in DISKI, Infix, Sankt Augustin, Germany

Hutter D (2000) Management of change in structured verification. In: Proceedings of Automated Software Engineering, ASE-2000, IEEE

Jamnik M, Kerber M, Pollet M, Benzmüller C (2003) Automatic learning of proof methods in proof planning. The Logic Journal of the IGPL 11(6):647–674

Kirchner H, Ringeissen C (eds) (2000) Frontiers of combining systems: Third International Workshop, FroCoS 2000, LNAI, vol 1794, Springer

Kohlhase M (2006) OMDOC - An Open Markup Format for Mathematical Documents [Version 1.2], LNAI, vol 4180. Springer

Lingenfelder C (1989) Structuring computer generated proofs. In: Proceedings of the International Joint Conference on AI (IJCAI'89), pp 378–383

Lingenfelder C (1990) Transformation and structuring of computer generated proofs. Doctoral thesis, University of Kaiserslautern, Department of Computer Science

Lusk E, Overbeek R (eds) (1988) Proceedings of the 9th Conference on Automated Deduction, no. 310 in LNCS, Springer, Argonne, Illinois, USA

Manthey R, Bry F (1988) SATCHMO: A theorem prover implemented in Prolog. In: (Lusk and Overbeek, 1988), pp 415–434

McCasland R, Bundy A (2006) MATHsAiD: a mathematical theorem discovery tool. In: Proceedings of SYNASC'06, IEEE Computer Society Press, pp 17–22

McCasland R, Bundy A, Smith P (2006) Ascertaining mathematical theorems. Electronic Notes in Theoretical Computer Science 151(1):21–38, proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2005)

McCune W (1997) Solution of the Robbins problem. Journal of Automated Reasoning 19(3):263–276

McCune WW (1994) Otter 3.0 reference manual and guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA

Meier A (2000) TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In: McAllester D (ed) Proceedings of the 17th Conference on Automated Deduction, Springer, no. 1831 in LNAI

Meier A (2004) Proof planning with multiple strategies. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany

Meier A, Melis E (2005) MULTI: A multi-strategy proof planner (system description). In: Nieuwenhuis R (ed) Proceedings of the 20^{th} Conference on Automated Deduction (CADE-20), Springer, Tallinn, Estonia, LNAI, vol 3632, pp 250–254

Meier A, Melis E, Pollet M (2002a) Towards extending domain representations. Seki Report SR-02-01, Department of Computer Science, Saarland University, Saarbrücken, Germany

Meier A, Pollet M, Sorge V (2002b) Comparing approaches to the exploration of the domain of residue classes. Journal of Symbolic Computation Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems 34:287–306

Melis E, Meier A (2000) Proof planning with multiple strategies. In: Loyd J, Dahl V, Furbach U, Kerber M, Lau K, Palamidessi C, Pereira L, Sagivand Y, Stuckey P (eds) First International Conference on Computational Logic (CL-2000), Springer, London, UK, no. 1861 in LNAI, pp 644–659

Melis E, Siekmann J (1999a) Knowledge-based proof planning. Artificial Intelligence 115(1):65–105

Melis E, Siekmann J (1999b) Knowledge-Based Proof Planning. Artificial Intelligence 115(1):65–105

Melis E, Zimmer J, Müller T (2000) Integrating constraint solving into proof planning. In: (Kirchner and Ringeissen, 2000)

Melis E, Meier A, Siekmann J (2007) Proof planning with multiple strategies. Artificial Intelligence

de Nivelle H (1999) Bliksem 1.10 user manual. Technical report, Max-Planck-Institut für Informatik

Riazanov A, Voronkov A (2001) Vampire 1.1 (system description). In: Goré R, Leitsch A, Nipkow T (eds) Automated Reasoning — 1st International Joint Conference, IJCAR 2001, Springer, no. 2083 in LNAI

Schönert M, et al (1995) GAP – Groups, Algorithms, and Programming. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany

Siekmann J, Hess S, Benzmüller C, Cheikhrouhou L, Fiedler A, Horacek H, Kohlhase M, Konrad K, Meier A, Melis E, Pollet M, Sorge V (1999) $\mathfrak{LOUI}$: $\mathfrak{L}$ovely ΩMEGA $\mathfrak{U}$ser $\mathfrak{I}$nterface. Formal Aspects of Computing 11:326–342

Siekmann J, Benzmüller C, Brezhnev V, Cheikhrouhou L, Fiedler A, Franke A, Horacek H, Kohlhase M, Meier A, Melis E, Moschner M, Normann I, Pollet M, Sorge V, Ullrich C, Wirth CP, Zimmer J (2002a) Proof development with ΩMEGA. In: (Voronkov, 2002), pp 143–148

Siekmann J, Benzmüller C, Fiedler A, Meier A, Pollet M (2002b) Proof development with OMEGA: Sqrt(2) is irrational. In: Baaz M, Voronkov A (eds) Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002, Springer, no. 2514 in LNAI, pp 367–387

Siekmann J, Benzmüller C, Fiedler A, Meier A, Normann I, Pollet M (2003) Proof development in OMEGA: The irrationality of square root of 2. In: Kamareddine F (ed) Thirty Five Years of Automating Mathematics, Kluwer Applied Logic series (28), Kluwer Academic Publishers, pp 271–314, iSBN 1-4020-1656-5

Siekmann J, Autexier S, Fiedler A, Gabbay D, Huang X (to appear) Principia Mathematica Mechanico, chap Proof Presentation

Sorge V (2000) Non-Trivial Computations in Proof Planning. In: (Kirchner and Ringeissen, 2000)

Sorge V (2001) ΩANTS — a blackboard architecture for the integration of reasoning techniques into proof planning. PhD thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany

Sutcliffe G, Zimmer J, Schulz S (2004) Tstp Data-Exchange Formats for Automated Theorem Proving Tools. In: Zhang W, Sorge V (eds) Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, pp 201–215

Voronkov A (ed) (2002) Proceedings of the 18th International Conference on Automated Deduction, no. 2392 in LNAI, Springer

Wagner M (2006) Mediation between text-editors and proof assistance systems. Diploma thesis, Saarland University, Saarbrücken, Germany

Wagner M, Autexier S, Benzmüller C (2006) Plato: A mediator between text-editors and proof assistance systems. In: Autexier S, Benzmüller C (eds) 7th Workshop on User Interfaces for Theorem Provers (UITP'06), Elsevier, ENTCS

Weidenbach C, Afshordel B, Brahm U, Cohrs C, Engel T, Keen E, Theobalt C, Topic D (1999) System description: SPASS version 1.0.0. In: (Ganzinger, 1999), pp 378–382

Wiedijk F (2004) Formal proof sketches. In: Berardi S, Coppo M, Damiani F (eds) Types for Proofs and Programs: Third International Workshop, TYPES 2003, Springer, Torino, Italy, LNCS 3085, pp 378–393

Wirth CP (2004) Descente infinie + Deduction. Logic J of the IGPL 12(1):1–96

Zhang J, Zhang H (1995) SEM: A system for enumerating models. In: Mellish CS (ed) Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann, San Mateo, California, USA, Montreal, Canada, pp 298–303

Zimmer J (2008) MATHSERVE – a framework for semantic reasoning services. PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany

Zimmer J, Autexier S (2006) The mathserve framework for semantic reasoning web services. In: Furbach U, Shankar N (eds) Proceedings of IJCAR'06, Springer, Seattle, USA, LNAI, pp 140–144

Zimmer J, Kohlhase M (2002) System description: The Mathweb Software Bus for distributed mathematical reasoning. In: (Voronkov, 2002), pp 138–142

Zimmer J, Melis E (2004) Constraint solving for proof planning. Journal of Automated Reasoning 33:51–88